

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

IN RE APPLICATION OF: Eun Hye CHOI, et al.

GAU:

SERIAL NO: New Application

EXAMINER:

FILED: Herewith

FOR: TRANSACTION PROCESSING SYSTEM SUPPORTING CONCURRENT ACCESSES TO
HIERARCHICAL DATA BY TRANSACTIONS

REQUEST FOR PRIORITY

COMMISSIONER FOR PATENTS
ALEXANDRIA, VIRGINIA 22313

SIR:

- ☐ Full benefit of the filing date of U.S. Application Serial Number , filed , is claimed pursuant to the provisions of 35 U.S.C. §120.
- ☐ Full benefit of the filing date(s) of U.S. Provisional Application(s) is claimed pursuant to the provisions of 35 U.S.C. §119(e): Application No. Date Filed

☒ Applicants claim any right to priority from any earlier filed applications to which they may be entitled pursuant to the provisions of 35 U.S.C. §119, as noted below.

In the matter of the above-identified application for patent, notice is hereby given that the applicants claim as priority:

COUNTRY

APPLICATION NUMBER

MONTH/DAY/YEAR

Japan

2003-025164

January 31, 2003

Certified copies of the corresponding Convention Application(s)

- ☒ are submitted herewith
- ☐ will be submitted prior to payment of the Final Fee
- ☐ were filed in prior application Serial No. filed
- ☐ were submitted to the International Bureau in PCT Application Number
Receipt of the certified copies by the International Bureau in a timely manner under PCT Rule 17.1(a) has been acknowledged as evidenced by the attached PCT/IB/304.
- ☐ (A) Application Serial No.(s) were filed in prior application Serial No. filed ; and
- ☐ (B) Application Serial No.(s)
- ☐ are submitted herewith
- ☐ will be submitted prior to payment of the Final Fee

Respectfully Submitted,

OBLON, SPIVAK, McCLELLAND,
MAIER & NEUSTADT, P.C.



Marvin J. Spivak

Registration No. 24,913

Customer Number

22850

Tel. (703) 413-3000
Fax. (703) 413-2220
(OSMMN 05/03)

C. Irvin McClelland
Registration Number 21,124

T656

日 本 国 特 許 庁
JAPAN PATENT OFFICE

別紙添付の書類に記載されている事項は下記の出願書類に記載されている事項と同一であることを証明する。

This is to certify that the annexed is a true copy of the following application as filed with this Office.

出 願 年 月 日 2 0 0 3 年 1 月 3 1 日
Date of Application:

出 願 番 号 特 願 2 0 0 3 - 0 2 5 1 6 4
Application Number:
[ST. 10/C]: [J P 2 0 0 3 - 0 2 5 1 6 4]

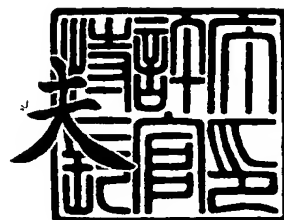
出 願 人 株 式 会 社 東 芝
Applicant(s):

3
J

2 0 0 3 年 7 月 1 8 日

特許庁長官
Commissioner,
Japan Patent Office

今 井 康



出証番号 出証特 2 0 0 3 - 3 0 5 7 4 9 2

【書類名】 特許願

【整理番号】 A000205442

【提出日】 平成15年 1月31日

【あて先】 特許庁長官 殿

【国際特許分類】 G06F 15/00

【発明の名称】 トランザクション処理システム、並行制御方法及びプログラム

【請求項の数】 18

【発明者】

【住所又は居所】 神奈川県川崎市幸区小向東芝町 1 番地 株式会社東芝研究開発センター内

【氏名】 崔 銀恵

【発明者】

【住所又は居所】 神奈川県川崎市幸区小向東芝町 1 番地 株式会社東芝研究開発センター内

【氏名】 金井 達徳

【特許出願人】

【識別番号】 000003078

【氏名又は名称】 株式会社 東芝

【代理人】

【識別番号】 100058479

【弁理士】

【氏名又は名称】 鈴江 武彦

【電話番号】 03-3502-3181

【選任した代理人】

【識別番号】 100091351

【弁理士】

【氏名又は名称】 河野 哲

【選任した代理人】

【識別番号】 100088683

【弁理士】

【氏名又は名称】 中村 誠

【選任した代理人】

【識別番号】 100108855

【弁理士】

【氏名又は名称】 蔵田 昌俊

【選任した代理人】

【識別番号】 100084618

【弁理士】

【氏名又は名称】 村松 貞男

【選任した代理人】

【識別番号】 100092196

【弁理士】

【氏名又は名称】 橋本 良郎

【手数料の表示】

【予納台帳番号】 011567

【納付金額】 21,000円

【提出物件の目録】

【物件名】 明細書 1

【物件名】 図面 1

【物件名】 要約書 1

【プルーフの要否】 要

【書類名】 明細書

【発明の名称】 トランザクション処理システム、並行制御方法及びプログラム

【特許請求の範囲】

【請求項 1】 階層型データを対象として複数のトランザクションを並列に処理するトランザクション処理システムにおける並行制御方法であって、

各トランザクションが前記階層型データへのアクセスを開始するにあたって、前記階層型データのコピーを作成するコピーステップと、

第 1 のトランザクションが該第 1 のトランザクション用の階層型データのコピーに対して読み出し又は書き込みの一方のアクセスを行う場合に、該アクセスと、第 2 のトランザクションが該第 2 のトランザクション用の階層型データのコピーに対して行った読み出し又は書き込みの他方のアクセスとの間に衝突が発生するか否かを判定する判定ステップと、

この判定ステップにおいて衝突が発生すると判定された場合に、衝突を回避するための処理を行う処理ステップと、

前記第 1 のトランザクションが正常に終了する場合に、該第 1 のトランザクションが該第 1 のトランザクション用の階層型データのコピーに行った書き込みアクセスを、前記階層型データに反映させるとともに、前記第 2 のトランザクションが未だ終了していないときは、該書き込みアクセスを、該第 2 のトランザクション用の階層型データのコピーにも反映させる反映ステップとを有することを特徴とする並行制御方法。

【請求項 2】 前記判定ステップでは、前記書き込みアクセスを考慮せずに前記読み出しアクセスを行って参照されるデータと、前記書き込みアクセスを考慮して前記読み出しアクセスを行って参照されるデータとの間の同一性に基いて、前記衝突が発生するか否かを判定することを特徴とする請求項 1 に記載の並行制御方法。

【請求項 3】 前記判定ステップでは、前記第 1 のトランザクションが前記階層型データのコピーに対して読み出しアクセスを行う場合には、該第 1 のトランザクション用の階層型データのコピーに対して該読み出しアクセスを行うとき

に参照される第1のデータと、該第1のトランザクション用の階層型データのコピーと前記第2のトランザクション用の階層型データのコピーとをマージしたもののに対して同じ読み出しアクセスを行うときに参照される第2のデータとの間の同一性を調べ、その結果に基づいて前記衝突が発生するか否かを判定することを特徴とする請求項1に記載の並行制御方法。

【請求項4】 すべての前記第2のトランザクションについて前記第1のデータと前記第2のデータとの間に同一性があると判断された場合に、前記衝突が発生しないと判定し、それ以外の場合に、前記衝突が発生すると判定することを特徴とする請求項3に記載の並行制御方法。

【請求項5】 前記第1のトランザクションが前記階層型データのコピーに対して書き込みアクセスを行う場合に、前記階層型データをアクセスするすべてのトランザクションによる書き込みアクセスを反映させるために前記階層型データをコピーして作成された共有コピーに対しても同じ書き込みアクセスを行うステップを更に有し、

前記判定ステップでは、前記第1のトランザクションが前記階層型データのコピーに対して読み出しアクセスを行う場合には、該読み出しアクセスによって参照される第1のデータと、前記階層型データの共有コピーに対して同じ読み出しアクセスを行うときに参照される第2のデータとの間の同一性を調べ、その結果に基づいて前記衝突が発生するか否かを判定することを特徴とする請求項1に記載の並行制御方法。

【請求項6】 前記第1のデータと前記第2のデータとの間に同一性があると判断された場合に、前記衝突が発生しないと判定し、同一性がないと判断された場合に、前記衝突が発生すると判定することを特徴とする請求項5に記載の並行制御方法。

【請求項7】 前記判定ステップでは、前記第1のトランザクションが前記階層型データのコピーに対して書き込みアクセスを行う場合には、前記階層型データをアクセスするすべての前記第2のトランザクションのすべての読み出しアクセスについて、当該読み出しアクセスを行ったときに参照された第1のデータと、該書き込みアクセスを行った後における前記階層型データの状態に対して当

該読み出しアクセスを行った当該第2のトランザクションを実行した場合に、当該読み出しアクセスを行うときに参照する第2のデータとの間の同一性を調べ、その結果に基づいて前記衝突が発生するか否かを判定することを特徴とする請求項1に記載の並行制御方法。

【請求項8】 前記階層型データをアクセスするすべての前記第2のトランザクションのすべての読み出しアクセスについて前記第1のデータと前記第2のデータとの間に同一性があると判断された場合に、前記衝突が発生しないと判定し、それ以外の場合に、前記衝突が発生すると判定することを特徴とする請求項7に記載の並行制御方法。

【請求項9】 前記階層型データをアクセスするすべてのトランザクションについて、各トランザクション毎に、当該トランザクションが前記階層型データのコピーに対して行ったアクセスの系列を記録するステップを更に有し、

前記判定ステップでは、前記アクセスの系列の記録を参照して、前記階層型データをアクセスするすべての前記第2のトランザクションのすべての読み出しアクセスを求めることを特徴とする請求項7または8に記載の並行制御方法。

【請求項10】 前記読み出しアクセスが行われたときに参照されたデータを記録するステップを更に有し、

前記判定ステップでは、前記参照されたデータの記録を参照して、前記第1のデータを求めることを特徴とする請求項7または8に記載の並行制御方法。

【請求項11】 前記判定ステップでは、前記読み出しアクセスを行ったトランザクションの開始時における前記階層型データの状態に対して該トランザクションが該読み出しアクセス以前に行った書き込みアクセスを行い、これによって得られる階層型データの状態に対して前記読み出しアクセスを行い、この結果得られるデータを、前記第1のデータとすることを特徴とする請求項7または8に記載の並行制御方法。

【請求項12】 前記第1のトランザクションが前記階層型データのコピーに対して書き込みアクセスを行う場合に、前記階層型データをアクセスするすべてのトランザクションによる書き込みアクセスを反映させるために前記階層型データをコピーして作成された共有コピーに対しても同じ書き込みアクセスを行う

ステップと、

前記階層型データをアクセスするいずれかのトランザクションが書き込みアクセスを行った時点における前記共有コピーの状態を保存するステップとを更に有し、

前記判定ステップでは、保存されている前記共有コピーの状態のうちから、前記読み出しアクセスを行った時点における前記階層型データの状態に近いものを選択するとともに、必要に応じて該共有コピーの状態に対して該読み出しアクセスを行ったトランザクションが行った書き込みアクセスを行って、該読み出しアクセスを行った時点における階層型データの状態を再現し、再現された該階層型データの状態に対して該読み出しアクセスを行い、この結果得られるデータを、前記第1のデータとすることを特徴とする請求項7または8に記載の並行制御方法。

【請求項13】 前記判定ステップでは、前記第1のトランザクション用の階層型データのコピーに対して前記書き込みアクセスを行った後の状態に対して、前記第2のトランザクションが前記読み出しアクセス以前に行った書き込みアクセスを行い、これによって得られる階層型データの状態に対して前記読み出しアクセスを行い、この結果得られるデータを、前記第2のデータとすることを特徴とする請求項7または8に記載の並行制御方法。

【請求項14】 前記第1のトランザクションが前記階層型データのコピーに対して書き込みアクセスを行う場合に、前記階層型データをアクセスするすべてのトランザクションによる書き込みアクセスを反映させるために前記階層型データをコピーして作成された共有コピーに対しても同じ書き込みアクセスを行うステップと、

前記階層型データをアクセスするいずれかのトランザクションが書き込みアクセスを行った時点における前記共有コピーの状態を保存するステップとを更に有し、

前記判定ステップでは、保存されている前記共有コピーの状態のうちから、前記読み出しアクセスを行いたい時点における前記階層型データの状態に近いものを選択し、該共有コピーの状態に対して、前記書き込みアクセスを行ったトランザ

クションがその時点以降に行った書込みアクセスを行うとともに、必要に応じて該読み出しアクセスを行ったトランザクションが行った書き込みアクセスを行って、該読み出しアクセスを行いたい時点における階層型データの状態を再現し、再現された該階層型データの状態に対して該読み出しアクセスを行い、この結果得られるデータを、前記第2のデータとすることを特徴とする請求項7または8に記載の並行制御方法。

【請求項15】 前記共有コピーの保存数に上限がある場合には、前記階層型データの各書き込みアクセス時の状態に対応する前記共有コピーのうち、後で読み出しアクセスを行いたい状態を再現するときに利用される可能性の高いものを優先して保存することを特徴とする請求項5、12または14に記載の並行制御方法。

【請求項16】 前記処理ステップでは、前記判定ステップにおいて衝突が発生すると判定された場合に、前記衝突に係るトランザクションのうち、所定の基準で決定されるものを、前記衝突に係る他のトランザクションが終了するまで待機させることを特徴とする請求項1ないし15のいずれか1項に記載の並行制御方法。

【請求項17】 階層型データを対象として複数のトランザクションを並列に処理するトランザクション処理システムであって、

各トランザクションが前記階層型データへのアクセスを開始するにあたって、前記階層型データのコピーを作成するコピー手段と、

第1のトランザクションが該第1のトランザクション用の階層型データのコピーに対して読み出し又は書き込みの一方のアクセスを行う場合に、該アクセスと、第2のトランザクションが該第2のトランザクション用の階層型データのコピーに対して行った読み出し又は書き込みの他方のアクセスとの間に衝突が発生するか否かを判定する判定手段と、

この判定手段において衝突が発生すると判定された場合に、衝突を回避するための処理を行う処理手段と、

前記第1のトランザクションが正常に終了する場合に、該第1のトランザクションが該第1のトランザクション用の階層型データのコピーに行った書き込みア

アクセスを、前記階層型データに反映させるとともに、前記第2のトランザクションが未だ終了していないときは、該書き込みアクセスを、該第2のトランザクション用の階層型データのコピーにも反映させる反映手段とを備えたことを特徴とするトランザクション処理システム。

【請求項18】 階層型データを対象として複数のトランザクションを並列に処理するトランザクション処理システムとしてコンピュータを機能させるためのプログラムであって、

各トランザクションが前記階層型データへのアクセスを開始するにあたって、前記階層型データのコピーを作成するコピー機能と、

第1のトランザクションが該第1のトランザクション用の階層型データのコピーに対して読み出し又は書き込みの一方のアクセスを行う場合に、該アクセスと、第2のトランザクションが該第2のトランザクション用の階層型データのコピーに対して行った読み出し又は書き込みの他方のアクセスとの間に衝突が発生するか否かを判定する判定機能と、

この判定機能において衝突が発生すると判定された場合に、衝突を回避するための処理を行う処理機能と、

前記第1のトランザクションが正常に終了する場合に、該第1のトランザクションが該第1のトランザクション用の階層型データのコピーに行った書き込みアクセスを、前記階層型データに反映させるとともに、前記第2のトランザクションが未だ終了していないときは、該書き込みアクセスを、該第2のトランザクション用の階層型データのコピーにも反映させる反映機能とをコンピュータに実現させるためのプログラム。

【発明の詳細な説明】

【0001】

【発明の属する技術分野】

本発明は、階層型データモデルに基づくデータベースを対象としたトランザクション処理システム、該トランザクション処理システムにおける並行制御方法及びプログラムに関する。

【0002】

【従来の技術】

トランザクション処理システムでは、トランザクションと呼ぶ処理の流れを単位として処理の実行を管理する。個々のトランザクションは、実行過程で、データベースのファイルに記録管理するデータにアクセスして、データの参照または更新を行う。

【0003】

一般にトランザクション処理システムでは、複数のトランザクションを並行に処理することで性能の向上を計る。その際、システムは、複数のトランザクションを並行に処理した場合の実行結果と、個々のトランザクションを1つずつ直列に処理した場合の実行結果とが同じであるように、トランザクションのアクセスを制御する必要がある。このことを、トランザクションの分離性(isolation)を保証する、あるいは、その実行は直列可能(serializable)であるという。

【0004】

トランザクションの分離性を保証するためには、並行に処理する複数のトランザクションが同じデータにアクセスすることを回避する必要がある。そのため、分離性の保証において扱いが難しいのは、同時に複数のトランザクションが1つのファイル上のデータにアクセスする場合である。複数のトランザクションが同じファイルにアクセスすることを許さなければこの問題は生じない。しかし、複数のトランザクションを並行に処理してシステムの性能を向上させるためには、1つのファイルの中の異なる部分に記録管理するデータに同時に複数のトランザクションがアクセスできるようにする必要がある。

【0005】

この問題を解決するために最も一般的に用いられている方式はロックである。ロック方式では、あるトランザクションがアクセスを行ったデータをそのトランザクションが終了するまでロックすることで、並行して処理する他のトランザクションが同じファイル上の同じ部分のデータにアクセスすることを回避し、同じファイル上の異なる部分のデータのみアクセスすることを可能にする。しかし、トランザクションの分離性を保証するロック方式を実現するためには、ファントム(phantom)と呼ばれる問題を解決しなければならない。

【0006】

ファントムとは、トランザクションがすでに削除したデータ、あるいは、これから挿入する可能性があるデータを表し、その時点においては存在しないデータである。例えば、あるトランザクションT1が条件Pを満たすデータを読み出した後に、並行に処理する他のトランザクションT2が条件Pを満たすようなあるデータを削除あるいは挿入したとする。トランザクションT2のアクセスによってデータが更新された後にトランザクションT1が条件Pを満たすデータの読出しを再度行ったときの結果は、トランザクションT2のアクセスの前にトランザクションT1が読出しを行ったときの結果と異なる。

【0007】

トランザクションの分離性を保証するためには、トランザクションT2が削除あるいは挿入したデータに対してロックを行い、並行に処理するトランザクションT1がファントムにアクセスできないようにする必要がある。しかし、ロックの対象であるデータはファントムであり、すでに削除されているか、あるいは、まだ挿入されていないためにロックの時点においては存在しない。したがって、ファントムはその扱いが困難である。

【0008】

ファントムの問題を解決する主なロック方式として、インデックスロック(index lock)、述語ロック(predicate lock)、プレジジョンロック(precision lock)の3つの方式が知られている。(例えば、非特許文献1参照)。

【0009】

1つ目のインデックスロックでは、データそのものではなく、データのインデックス(index)をロックの対象とする。インデックスはデータの検索を高速にするために用いるデータの値に基づいた索引であり、インデックスの構造としてB-T r e e、ハッシュテーブルなどの種類が知られている。インデックスロックでは、インデックスの構造を利用して、ファントムを参照する可能性があるインデックスの範囲をロックすることによってファントムの問題を解決し、トランザクションの分離性を保証する。

【0010】

2つ目の述語ロックでは、データそのものではなく、データの集合を特定する述語をロックの対象とすることで、ファントムの問題を解決する。通常、トランザクションが行うデータへのアクセスは、そのデータを特定する述語によって行われる。述語ロックでは、あるトランザクションがアクセスに使用した述語をロックし、他のトランザクションがアクセスに使用する述語とすでにロックした述語を比較することで、トランザクションの分離性が破られないかを検査する。

【0011】

3つ目のプレジジョンロックは、この述語ロックを改善した方式であり、述語ロックと同様にファントムの問題を解決できる。この方式の特徴は、トランザクションがデータへのアクセスを要求したときに、他のトランザクションがすでに行ったアクセスで使用した述語とそのデータを比較することである。比較の結果、データが述語を充足しなければトランザクションの分離性は維持される。

【0012】

【非特許文献1】

“Transaction Processing: Concepts and Techniques” (Jim Gray, Andreas Reuter著, Morgan Kaufmann, 1993)

【0013】

【発明が解決しようとする課題】

トランザクション処理の対象となるデータの集合あるいはファイルを管理する方式として、従来はリレーショナルデータモデルに基づく関係データベースが主流であったが、近年では階層型モデルのデータを管理するデータベースの必要性が高まっている。階層型データモデルの例としては、インターネット上で交換するデータの標準フォーマットとして注目されているXMLがある。

【0014】

ここで、階層型データモデルに基づくデータベースを対象としてトランザクション処理を行う場合について、従来の3つのロック方式、すなわち、インデックスロック、述語ロック、プレジジョンロックのそれぞれが抱える問題点について述べる。

【0015】

まず、インデックスロックでは、データのファイルから導出するインデックス構造を使用する。リレーショナルデータモデルに対してはB-T r e eなどの有効なインデックス構造が知られており、従来のほとんどの関係データベースはインデックスロックに基づく方式を採用している。しかし、階層型データモデルでは、データの親子関係がツリー構造で表現される、あるいは、データの重複が許されるなどといった理由から、有効なインデックス構造を導出できない。この問題を解決するために、階層型データモデルをリレーショナルデータモデルに変換して関係データベースとして管理する方式もある。しかし、そのような方式は、データのファイルが持つ本来の階層構造を効率よく管理できず、かつ、すべての階層型データモデルに対して有効ではないといった問題点がある。そのため、階層型データモデルに基づくデータベースに対してインデックスロックを使用するのは困難である。

【0016】

述語ロックでは、トランザクションの分離性を検査するために述語同士の比較を行う必要がある。一般に述語の充足性判定はNP完全であることが知られており、述語ロックの実装には大変コストがかかる。

【0017】

述語ロックを改善した方式であるプレジジョンロックでは、述語同士の比較を行う代わりにデータと述語の比較を行うため、述語ロックに比べてコストが小さい。また、トランザクションがアクセスに使用した述語を予めロックする方法ではなく、アクセスが要求された時点において分離性の検査を行う方法を用いるので、トランザクションの並行処理性に優れている。しかし、インデックスロックと比べるとコストが高いという問題点があり、関係データベースが主流であった従来では、インデックスロックに基づく方式が主に使用されていた。さらに、プレジジョンロックについては、概念のみが知られており、実装方式は提案されていない。プレジジョンロックを階層型データモデルに適用するためには、トランザクションがアクセスして更新しようとする階層型データが、並行に処理する他のトランザクションのアクセス時にすでに使用された述語を充足するか否かの判定をして分離性の検査を行う必要がある。しかし、このような問題を解決する実

用的な方式は、まだ提案されていない。

【0018】

現状では、XMLデータのような階層型データモデルに基づくデータベースに対してトランザクションの分離性を保証するためには、並行に処理するトランザクションがアクセスするデータファイル全体をロックするといった方式が使用されている。

【0019】

本発明は、上記事情を考慮してなされたもので、階層型データを複数のトランザクションが並行してアクセスする場合にも、トランザクションの分離性を保証することができる、あるいは、その実行が直列化可能であるように処理の順序を制御することができるトランザクション処理システム、並行制御方法及びプログラムを提供することを目的とする。

【0020】

【課題を解決するための手段】

本発明は、階層型データを対象として複数のトランザクションを並列に処理するトランザクション処理システムにおける並行制御方法であって、各トランザクションが前記階層型データへのアクセスを開始するにあたって、前記階層型データのコピーを作成するコピーステップと、第1のトランザクションが該第1のトランザクション用の階層型データのコピーに対して読み出し又は書き込みの一方のアクセスを行う場合に、該アクセスと、第2のトランザクションが該第2のトランザクション用の階層型データのコピーに対して行った読み出し又は書き込みの他方のアクセスとの間に衝突が発生するか否かを判定する判定ステップと、この判定ステップにおいて衝突が発生すると判定された場合に、衝突を回避するための処理を行う処理ステップと、前記第1のトランザクションが正常に終了する場合に、該第1のトランザクションが該第1のトランザクション用の階層型データのコピーに行った書き込みアクセスを、前記階層型データに反映させるとともに、前記第2のトランザクションが未だ終了していないときは、該書き込みアクセスを、該第2のトランザクション用の階層型データのコピーにも反映させる反映ステップとを有することを特徴とする。

【0021】

また、本発明は、階層型データを対象として複数のトランザクションを並列に処理するトランザクション処理システムであって、各トランザクションが前記階層型データへのアクセスを開始するにあたって、前記階層型データのコピーを作成するコピー手段と、第1のトランザクションが該第1のトランザクション用の階層型データのコピーに対して読み出し又は書き込みの一方のアクセスを行う場合に、該アクセスと、第2のトランザクションが該第2のトランザクション用の階層型データのコピーに対して行った読み出し又は書き込みの他方のアクセスとの間に衝突が発生するか否かを判定する判定手段と、この判定手段において衝突が発生すると判定された場合に、衝突を回避するための処理を行う処理手段と、前記第1のトランザクションが正常に終了する場合に、該第1のトランザクションが該第1のトランザクション用の階層型データのコピーに行った書き込みアクセスを、前記階層型データに反映させるとともに、前記第2のトランザクションが未だ終了していないときは、該書き込みアクセスを、該第2のトランザクション用の階層型データのコピーにも反映させる反映手段とを備えたことを特徴とする。

【0022】

また、本発明は、階層型データを対象として複数のトランザクションを並列に処理するトランザクション処理システムとしてコンピュータを機能させるためのプログラムであって、各トランザクションが前記階層型データへのアクセスを開始するにあたって、前記階層型データのコピーを作成するコピー機能と、第1のトランザクションが該第1のトランザクション用の階層型データのコピーに対して読み出し又は書き込みの一方のアクセスを行う場合に、該アクセスと、第2のトランザクションが該第2のトランザクション用の階層型データのコピーに対して行った読み出し又は書き込みの他方のアクセスとの間に衝突が発生するか否かを判定する判定機能と、この判定機能において衝突が発生すると判定された場合に、衝突を回避するための処理を行う処理機能と、前記第1のトランザクションが正常に終了する場合に、該第1のトランザクションが該第1のトランザクション用の階層型データのコピーに行った書き込みアクセスを、前記階層型データに

反映させるとともに、前記第2のトランザクションが未だ終了していないときは、該書き込みアクセスを、該第2のトランザクション用の階層型データのコピーにも反映させる反映機能とをコンピュータに実現させるためのプログラムである。

【0023】

なお、装置に係る本発明は方法に係る発明としても成立し、方法に係る本発明は装置に係る発明としても成立する。

また、装置または方法に係る本発明は、コンピュータに当該発明に相当する手順を実行させるための（あるいはコンピュータを当該発明に相当する手段として機能させるための、あるいはコンピュータに当該発明に相当する機能を実現させるための）プログラムとしても成立し、該プログラムを記録したコンピュータ読取り可能な記録媒体としても成立する。

【0024】

本発明によれば、例えばXMLのような階層型データを複数のトランザクションが並行してアクセスする場合にも、トランザクションの分離性を保証することができる、あるいは、その実行が直列化可能であるように処理の順序を制御することができるようになる。

【0025】

【発明の実施の形態】

以下、図面を参照しながら発明の実施の形態を説明する。

【0026】

図1に、本発明の一実施形態に係るトランザクション処理システムの構成例を示す。図中、1はトランザクション管理部、11はトランザクションマネージャ、12はリソースマネージャ、111はトランザクション管理表、3はハードディスク、31はファイル、5はアプリケーションプログラムをそれぞれ示している。

【0027】

なお、トランザクション処理システムがハードディスク3を備えてもよいし、他のサーバ等がハードディスク3を備え、トランザクション処理システムが他の

サーバ等を介してハードディスク 3 にアクセス可能であってもよい。また、アプリケーションプログラム 5 はトランザクション処理システム上で実行するものであってもよいし、アプリケーションプログラム 5 は他の計算機上で実行されるもので、他の計算機をクライアント、トランザクション処理システムをサーバとする、クライアント・サーバ・システムであってもよい。

【0028】

図 1 のハードディスク 3 には、トランザクションのアクセスの対象となるデータのファイル 31 が記録されている。ここでは、階層型データモデルの一例である XML 形式のドキュメントとしてデータが記録されているファイルを対象とするトランザクション処理システムを例にとって説明する。XML に関しては、“Extensible Markup Language (XML) 1.0” (W3C Recommendation 10-Feb-1998) に開示されている。

【0029】

ハードディスク 3 に記録するファイル 31 のドキュメント形式は、テキスト形式であってもツリー形式であっても構わない。図 2 は、テキスト形式の XML ドキュメントの例を示している。実際の XML ドキュメントには `<?xml>` で始まるプロローグ部があるが、ここでは省略する。図 3 は、図 2 と同じデータをツリー形式で表現しているドキュメントの例である。

【0030】

図 2 のテキスト形式のドキュメントは、`<flowers>` と `</flowers>` のタグで囲まれている。テキスト形式のドキュメントを囲む一番外側のタグは、ツリー形式のドキュメントにおけるツリーの根に相当する。例えば、図 3 のツリー形式のドキュメントでは、`flowers` という名前のノードがツリーの根となっている。

【0031】

データの階層関係は、テキスト形式のドキュメントではタグの入れ子関係によって、ツリー形式のドキュメントではノードの親子関係によって表現される。例えば、図 2 の `<flowers>` と `</flowers>` のタグの内側には `<flower>` と `</flower>` の入れ子タグが 3 つあり、図 3 のツリーの根

ノードの下には名前が `flower` である子ノードが3つある。図2のドキュメントの `flower` タグの内側にはさらに `name`、`color`、`price` の3つのタグがあり、例えば、1つ目の `flower` タグの内側の `name` タグで囲まれたデータは “`Tulip`” となっている。これに対して、図3のドキュメントでは、1つ目の `flower` ノードの子である `name` ノードのさらに子であるツリーの葉ノードの名前が “`Tulip`” となっていてデータの値を表している。

【0032】

以下では、各ファイルはツリー形式のドキュメントとして記録されている場合を例にとって説明するが、各ファイルがテキスト形式で記録されている場合でも例えばツリー構造への変換を加えることなどによって同様に実施できる。

【0033】

図1のアプリケーションプログラム5は、ハードディスク3に記録されているファイル31にアクセスしてデータの操作（読出しあるいは書込み）を行う。そのために、トランザクションを発行し、トランザクション管理部1を介してトランザクションの処理を行う。

【0034】

本実施形態の並列制御方式が扱う問題は、同じファイルにアクセスする複数のトランザクションの並行処理を、分離性を維持しながら行う問題である。以降の本実施形態の並行制御方式の説明では、トランザクションはその実行過程で1つのファイルのみにアクセスする場合を中心に説明する。通常のトランザクションは複数のファイルにアクセスしてデータの操作することもできるが、トランザクションのアクセスの対象となる個々のファイルごとにその処理を分けて考えることで、1つのトランザクションが複数のファイルにアクセスする場合も同様に実施することができる。

【0035】

トランザクションのアクセスには、データの参照を行うための読出しアクセスとデータの更新（例えば、挿入、削除、値の変更）を行うための書込みアクセスとがある。本実施形態におけるトランザクションは、1つのファイルに対して行

う1つ又は複数の読出しアクセス及び又は書込みアクセスのアクセス系列からなる。

【0036】

まず、トランザクションの読出しアクセスは、`READ (path)` という操作を行う。一般に階層型データモデルでは、パス形式の述語を用いることによって参照するデータ（ツリー形式においてはデータに対応するノード）を指定できる。例えば、XMLドキュメント上のある部分のデータあるいはデータの集合を指定するためには、`XPath` というパス形式の言語がよく用いられる。`XPath` に関しては、“XML Path Language (XPath) 1.0” (W3C Recommendation 16-Nov-1999)に開示されている。`READ (path)` における `path` は、例えば、`XPath` のようなパス形式の述語である。`READ (path)` は、`path` によって指定されたドキュメント上のノードあるいはノードの集合を返す操作である。

【0037】

トランザクションは、`READ (path)` の結果として返されるノードから参照したいデータの値を読み出すこととなる。例えば、`path = “flower [name=Tulip] / color”` は、`XPath` 言語を用いた一例であり、`[name=Tulip]` の条件を満たす `flower` の子ノード `color` を指定する述語である。トランザクションが図3のドキュメントに対する読出しアクセスとして `READ (“flower [name=Tulip] / color”)` の操作を行うと、その結果として図3のノード `n5` が返される。トランザクションは、ノード `n5` の値（ツリーにおいては子ノードの名前）から “Yellow” を読み出すことができる。もう1つの例として、トランザクションが図3のドキュメントに対して `READ (“flower [price<400] / name”)` を行うと、この場合は、ノードの集合 {ノード `n4`, ノード `n10`} が返される。トランザクションは、それぞれのノードの値からデータ “Tulip” と “Lilac” を読み出すことができる。

【0038】

トランザクションの書込みアクセスでは、ここでは、`INSERT`、`DELETE`

TE、REPLACEの3種類の操作を行うことができるものとする。なお、ここでは、トランザクションの書込みアクセスの操作として上記の3つのみを挙げたが、ドキュメントのノードに対して更新を行う他の操作も可能であり、その場合でも本実施形態の並列制御方式を同様に実施することができる。

【0039】

以下、INSERT操作、DELETE操作、REPLACE操作についてそれぞれ説明する。

【0040】

INSERT (node, data) は、nodeで指定したノードの値に、dataで指定した値を挿入する操作である。例えば、図4のドキュメントに対する書込みアクセスとしてINSERT (ノードn5, “Yellow”) の操作を行うと、その更新結果を反映したドキュメントは図3のようになる。

【0041】

INSERT (node, child-node) は、nodeで指定したノードの子ノードとして、child-nodeで指定したノードを挿入する操作である。この操作の他にも、例えば、何番目の子ノードとして挿入するかを指定する操作、兄弟ノードとしてnodeで指定したノードの前に挿入する操作、兄弟ノードとしてnodeで指定したノードの後に挿入する操作など、さまざまなINSERT操作を考えてもよい。

【0042】

DELETE (node) は、nodeで指定したノードを削除する操作である。例えば、図3のドキュメントに対する書込みアクセスとしてDELETE (ノードn13) の操作を行うと、その更新結果を反映したドキュメントは図4のようになる。

【0043】

REPLACE (node, data) は、nodeで指定したノードの値をdataで指定した値に変更する操作である。例えば、図3のドキュメントに対する書込みアクセスとしてREPLACE (ノードn5, “Red”) の操作を行うと、その更新結果を反映したドキュメントは図5のようになる。

【0044】

これら INSERT、DELETE、REPLACE とは異なる操作を考慮する場合でも同様に実施することが可能である。例えば、XML ドキュメントのノードには属性を与えることができる。この場合には、書込みアクセスとして、INSERT (node, attr, value) といった node で指定したノードの attr という名前の属性に value で指定した値を挿入する操作を加えてもよい。

【0045】

ところで、トランザクションがデータの更新を行うときは、読出しアクセスによって更新したいデータを指定した後に、そのデータに対して書込みアクセスの操作を行う必要がある。すなわち、書込みアクセスは、読出しアクセスの後に、その読出しアクセスの結果として返されるノードあるいはノードの集合に対して行われる。例えば、トランザクションが図3のドキュメントで Tulip の color の値を “Red” に更新する場合には、まず、読出しアクセスの READ (“flower [name=Tulip] / color”) を行った後に、その結果として返されるノード n5 に対して、書込みアクセスの REPLACE (ノード n5, “Red”) を行う。

【0046】

なお、ここでは1つのノードに対して書込みアクセスの操作を行う場合の例を説明したが、ノードの集合を対象とする場合も個々のノードに対する更新を行うことで同様に実施できる。

【0047】

図1のトランザクション管理部1は、各アプリケーションプログラム5が実行するトランザクションの処理を行う。トランザクション管理部1は、トランザクションマネージャ11とリソースマネージャ12とを含んでいる。トランザクションマネージャ11は、アプリケーションプログラム5から発行されるすべてのトランザクションの管理を行う。他方、リソースマネージャ12は、データベース上のファイル31の管理と、そのファイル31に対して各トランザクションが行うアクセスの処理を行う。

【0048】

図1では、トランザクション管理部1が複数のリソースマネージャ12を含んでいる場合を例にとって示している。ここでは、各リソースマネージャ12は、データベース上の個々のファイル31を担当しており、担当するファイル31に対するトランザクションのアクセスを処理するものとしている。もちろん、これに限定されるものではなく、他の構成をとっても構わない。例えば、1つのリソースマネージャ12がすべてのファイル31へのトランザクションのアクセスを処理するようにして、1つのトランザクションマネージャ11と1つのリソースマネージャ12を含むトランザクション管理部1として実施することもできるし、少なくとも1つのリソースマネージャ12が複数のファイル31へのトランザクションのアクセスを処理するようにして、1つのトランザクションマネージャ11と複数(図1よりは少ない数)のリソースマネージャ12を含むトランザクション管理部1として実施することもできる。

【0049】

図1のトランザクションマネージャ11は、アプリケーションプログラム5が発行するすべてのトランザクションを管理する。また、アプリケーションプログラム5が発行した個々のトランザクションを、そのトランザクションがアクセスするファイル31を管理しているリソースマネージャ12に対応付ける。そして、各トランザクションのアクセスの処理を、対応するリソースマネージャ12に指示する。トランザクションマネージャ11は、新しいファイル31の作成および消去に応じて、リソースマネージャ12の作成および削除を行う。

【0050】

図1のトランザクション管理表111は、どのトランザクションがどのリソースマネージャ12に対応しているかを管理する。トランザクション管理表111には、トランザクションのトランザクション識別子と、そのトランザクションがアクセスするファイル31を管理しているリソースマネージャ12の識別子との対応を示す情報が記録されている。例えば、図6のトランザクション管理表の例では、トランザクション識別子T1, T3, T5の3つのトランザクションが、リソースマネージャR1が管理しているファイル31にアクセスしていることを

示している。

【0051】

以降では、各トランザクションは1つのファイル31に対してアクセスを行い、そのファイル31を管理している1つのリソースマネージャ12に対応している場合を例としての処理手順を説明する。本実施形態の並行制御方式は、各ファイル31に対するトランザクションのアクセスを処理する個々のリソースマネージャによって実施されるので、各トランザクションが複数のリソースマネージャに対応する場合においても同様に実施できる。

【0052】

以下では、トランザクションマネージャ11が行う処理手順について、(1) トランザクションが発行されたときの処理手順、(2) トランザクションが読出しアクセスあるいは書込みアクセスを要求したとき処理手順、(3) トランザクションの処理を終了するときの処理手順の順番に説明する。

【0053】

(1) トランザクションが発行されたときの処理手順

アプリケーションプログラム5が新しいトランザクションの発行と、そのトランザクションがアクセスするファイル31とを、トランザクションマネージャ11に知らせると、トランザクションマネージャ11は、まず、新しいトランザクションにトランザクション識別子を割り付ける。また、そのトランザクションがアクセスするファイル31を管理しているリソースマネージャ12を調べ、トランザクション識別子と対応するリソースマネージャ12識別子の情報をトランザクション管理表111に記録する。次に、対応するリソースマネージャ12に対して、新しいトランザクションの処理開始を指示する。

【0054】

(2) トランザクションがアクセスを要求したときの処理手順

アプリケーションプログラム5がトランザクションの実行を進めていく過程で読出しアクセスあるいは書込みアクセスを要求すると、トランザクションマネージャ11は、対応するリソースマネージャ12に、トランザクション識別子とそのトランザクションのアクセス要求とを知らせる。

【0055】

(3) トランザクションの処理を終了するときの処理手順

アプリケーションプログラム 5 がトランザクションの処理を終了すると知らせると、トランザクションマネージャ 11 は、トランザクション識別子からそのトランザクションを処理しているリソースマネージャ 12 を調べて、対応するリソースマネージャ 12 に、トランザクションが行ったデータの更新結果をハードディスク 3 上のファイル 31 に書き込んでコミットするか、または更新結果を破棄してアボートするかを決定し、指示する。なお、コミットするかアボートするかは、従来通りで構わない。また、トランザクション管理表 111 から、そのトランザクション識別子のエントリを削除する。

【0056】

図 1 のリソースマネージャ 12 は、対応するハードディスク 3 上のファイル 31 を管理し、トランザクションマネージャ 11 から指示されたときにトランザクションのアクセスを処理する。トランザクションのアクセスを処理する際には、本実施形態の並行制御方式を実施し、トランザクションの分離性を維持するように処理する。

【0057】

本実施形態の並行制御方式では、あるトランザクションが読出しアクセスあるいは書込みアクセスを要求したときに、そのアクセスが、そのトランザクションと並行に処理中の他のトランザクションが行った読出しアクセスあるいは書込みアクセスと同じファイルの同じ部分のデータに対して行われることで、トランザクションの分離性を破らないかを検査する。

【0058】

ここで、2つのアクセスが同じファイルの同じ部分のデータにアクセスすることを、2つのアクセスは衝突するという。

【0059】

読出しアクセスと読出しアクセスとの間の衝突では、同時に同じ部分のデータを読み出しても分離性を破らないので、この検査の必要はない。

【0060】

他方、読出しアクセスと書込みアクセスとの間の衝突では、同時に同じ部分のデータの読出しと書き込みを行うと分離性を破ってしまうので、この検査を行う必要がある。

【0061】

同様に、書込みアクセスと書込みアクセスとの衝突も分離性を破る。ただし、あるデータに対する書込みアクセスを行う前にはかならずそのデータに対する読出しアクセスが行われるので、分離性を破る書込みアクセスと書込みアクセスとの衝突は、読出しアクセスと書き込みアクセスとの衝突を検査することで発見できることになり、結局、この検査を行う必要はないことになる。

【0062】

アクセス衝突の検査の結果、あるトランザクションT1が要求したアクセスが、他のトランザクションT2がすでに行ったアクセスと衝突を起こして分離性を破る場合は、いずれか一方のトランザクションの処理が終了するまで、他方のトランザクションの処理を中断しなければならない。その際は、いずれのトランザクションの処理を優先するかによって、いずれのトランザクションを中断すべきか決めればよい。例えば、先のトランザクションT2を優先し、後からのトランザクションT1を中断して、トランザクションT2が終了した後にトランザクションT1を再開する方法や、予めトランザクションごとに優先度を付与しておいて、衝突を起こしたトランザクションの優先度を比較することによって、いずれのトランザクションの処理を優先するかを決定するようにしてもよいし、その他にも、種々の方法が可能である。

【0063】

本実施形態では、個々のリソースマネージャが管理しているファイルに対するトランザクションのアクセスを処理する際に、アクセス衝突の検査を行う。本実施形態で用いるアクセス衝突の検査方法については後で詳しく説明する。

【0064】

以下、本実施形態の並行制御を実施するリソースマネージャとして2つの構成例を示す。

【0065】

(リソースマネージャの第1の構成例)

まず、リソースマネージャの第1の構成例について説明する。

【0066】

図7に、第1の構成例に係るリソースマネージャの構成例を示す。図中、12はリソースマネージャ、121はドキュメントD-a11、122はトランザクション待ちグラフ、123はトランザクションリスト、124はトランザクションアクセス系列、125はドキュメントD(Tid)を、3はハードディスクを、31はドキュメントD-st(図1のファイル31)をそれぞれ示している。

【0067】

リソースマネージャ12は、1つのファイルを管理し、そのファイルに対する複数のトランザクションのアクセスを処理する。図7のドキュメントD-stは、リソースマネージャ12が管理するハードディスク3上のファイル(図1の31)を指す。以降で説明するドキュメントは、すべてファイルD-stと同じツリー形式のドキュメントである。

【0068】

図7のドキュメントD-a11(図中の121)は、リソースマネージャ12が管理するファイル31に対して、処理中のすべてのトランザクションが現在までに行ったデータの更新結果を反映させた場合の内容を保持させるドキュメントである。リソースマネージャ12は、初期状態でドキュメントD-stをコピーしてドキュメントD-a11を作成する。以降、リソースマネージャ12は、処理中のトランザクションが要求する書込みアクセスが分離性を破らないと判断すれば、その書込みアクセスによるデータの更新をドキュメントD-a11に重ねて反映させていく。

【0069】

図7のトランザクションリスト123には、リソースマネージャ12が処理しているトランザクションのトランザクション識別子のリストが記録管理されている。例えば、図8のトランザクションリストの例では、図6の例における識別子R1のリソースマネージャ12が、トランザクション識別子T1、T3、T5の3つのトランザクションの処理を並行に行っていることを示している。リソース

マネージャ 12 は、処理中の個々のトランザクション識別子 Tid のトランザクションに対して、トランザクションアクセス系列 $AS(Tid)$ (図中の 124) とドキュメント $D(Tid)$ (図中の 125) を管理する。

【0070】

図 7 のトランザクション待ちグラフ 122 は、リソースマネージャ 12 が処理を中断して待機させているトランザクションのトランザクション識別子の情報を記録管理する待ちグラフである。待ちグラフの各点はトランザクションを表し、各辺はトランザクションの実行順序の依存関係を表す。

【0071】

図 9 は、トランザクション待ちグラフの例を示している。例えば、図 9 の辺 ($T3 \rightarrow T1$) は、トランザクション $T3$ の処理が終了するまでトランザクション $T1$ が処理を中断して待機していることを表す。もしトランザクション $T1$ のアクセスがトランザクション $T3$ のアクセスと衝突するときは、トランザクション $T3$ の処理が終了するまでトランザクション $T1$ を待機させなければならないので、リソースマネージャ 12 は、トランザクション待ちグラフ 122 に辺 ($T3 \rightarrow T1$) を追加する。そして、トランザクション $T3$ の処理を終了するときに、 $T3$ が始点である辺を削除し、その辺の終点のトランザクションの処理を再開する。

【0072】

トランザクション待ちグラフ 122 は、デッドロックを解決するためにも広く用いられる。待ちグラフの中にループがあるとデッドロックの状態であることがわかる。

【0073】

本実施形態では、トランザクションの待ち情報を記録管理するために待ちグラフを使用するが、他の方法を使用して実施することも可能である。

【0074】

図 7 のトランザクションアクセス系列 $AS(Tid)$ (図中の 124) は、個々のトランザクション Tid について、それが処理の開始から行ってきた読出しアクセスおよび書込みアクセスの系列を、リストとして記録管理している。トラ

ンザクションアクセス系列 124 には、各アクセスについて順番に、当該アクセスのアクセス番号と、当該アクセスが読出しアクセス、書込みアクセスのいずれであるかを示す情報と、当該アクセスの操作を示す情報とが記録されている。A S (T i d) は、トランザクション識別子 T i d のトランザクションのトランザクションアクセス系列を指す。

【0075】

図 10 は、トランザクションアクセス系列の一例を示している。図 10 における r は読出しアクセスであることを、w は書込みアクセスであることをそれぞれ表す。図 10 の例のトランザクションアクセス系列を持つトランザクションは、最初にアクセス番号 1 の読出しアクセス READ (“flower / name”) を行い、次にアクセス番号 2 の読出しアクセス READ (“flower [name = Tulip] / color”) を行い、最後にアクセス番号 3 の書込みアクセス REPLACE (node₂, “Red”) を行っている。ここで node₂ はアクセス番号 2 の読出しアクセスの結果として返されたノードを指す。書込みアクセスを行うときの操作対象のノードとしては、以前に行った読出しアクセスの結果のノードおよびノードの集合、あるいはその一部のノードなどを指定することができる。

【0076】

図 7 のドキュメント D (Tid) (図中の 125) は、トランザクション識別子 Tid のトランザクションが行った書込みアクセスによるデータの更新結果を反映しているドキュメントである。なお、以降の説明では、トランザクション識別子 Tid を省略して、(当該トランザクションに対応する) ドキュメント D と呼ぶこともある。ドキュメント D - a l l がリソースマネージャ 12 の処理するすべてのトランザクションにより行われたデータの更新を反映しているのに対して、このドキュメント D は対応する 1 つのトランザクションが行ったデータの更新を反映している。

【0077】

リソースマネージャ 12 は、トランザクションの処理を開始するときに、管理するファイル、すなわち、ドキュメント D - s t をコピーして新しいトランザク

ション用のドキュメントDを作成する。以降、そのトランザクションが読出しアクセスあるいは書込みアクセスを要求したときには、そのアクセスが分離性を破らないと判断すれば、ハードディスク3上のドキュメントD-s tの代わりにドキュメントDにアクセスしてデータの参照あるいは更新を行う。そして、そのトランザクションをコミットして終了するときに、そのトランザクションに対応するドキュメントDに対して行った更新を、ドキュメントD-s tにマージすることによって、コミットするデータの更新結果をハードディスク3上のファイル3.1に反映する。他方、トランザクションをアボートして終了するときには、そのトランザクションによる更新結果を破棄するので、ドキュメントDを削除すればよい。

【0078】

リソースマネージャ12は、トランザクションからアクセスを要求されたとき、そのアクセスが分離性を破らないかを判断しなければならない。トランザクションが読出しアクセスを要求したときは、並行して処理中の他のトランザクションがすでに書込みアクセスを行ったデータと同じ部分にアクセスして衝突を起こさないかを検査する。また、トランザクションが書込みアクセスを要求したときは、並行して処理中の他のトランザクションがすでに読出しアクセスを行ったデータと同じ部分にアクセスして衝突を起こさないかを検査する。以降、読出しアクセスがすでに行われた書込みアクセスと衝突することを「RWアクセス衝突」、書込みアクセスがすでに行われた読出しアクセスと衝突することを「WRアクセス衝突」とそれぞれ呼ぶものとする。

【0079】

(RWアクセス衝突の検査)

まず、RWアクセス衝突の検査について説明する。

【0080】

RWアクセス衝突は、あるトランザクションT1が要求する読出しアクセスが、並行して処理中の他のトランザクションがすでに行った書込みアクセスに対して衝突を起こすことである。

【0081】

従来の述語ロックやプレジジョンロックでは、述語と述語の比較や述語とデータの比較によって、この衝突を検出する。しかし、トランザクションT1の読出しアクセスREAD (path) におけるXPath式のpathを、他のトランザクションがすでに更新したデータが充足するか否かを判定するのは、非常に難しい問題である。

【0082】

本実施形態の並行制御方式では、データとデータとの比較のみによってアクセス衝突の検査を効率よく実現する。

【0083】

まず、あるトランザクションT1が要求する読出しアクセスが、他のトランザクションT2がすでに行った書込みアクセスに対してRWアクセス衝突を起こす場合を考える。この場合、トランザクションT1の要求する読出しアクセスが、トランザクションT2がすでに更新したデータと同じ部分のデータを参照するので、アクセス衝突が起きる。

【0084】

トランザクションT1が読出しアクセスを行うときには、ドキュメントD (T1) に対して読出し操作READ (path) が行われ、pathによって特定されるドキュメントD (T1) 上のノードの集合が、読出しアクセスの結果として返される。XPath式を評価して結果のノード集合を特定するためには、ツリー構造であるドキュメントD (T1) 上の経路を、pathの記述に従ってたどりながら、該当するノードを探索する必要があり、探索経路の最後のステップにおいて結果のノード集合が得られる。したがって、読出しアクセスは、結果のノード集合に至る経路上のすべてのノードを参照することとなる。トランザクションT1の読出しアクセスが参照するこれらのノードの集合をN1とする。

【0085】

トランザクションT2がすでに行った書込みアクセスの更新結果は、ドキュメントD (T2) に反映されている。ドキュメントD (T1) にトランザクションT2がすでに行った更新結果を反映したドキュメントは、ドキュメントD (T1) とD (T2) とをマージして得られる。ここで、2つのドキュメントD (T1

)とD(T2)をマージするということは、トランザクションT1とT2がそれぞれ行った更新結果の両方がマージしたドキュメントに反映されているということの意味する。マージしたドキュメントに対して同様に読出しアクセスREAD(path)を行ったときに参照するノードの集合をN2とする。

【0086】

RWアクセス衝突が起きるとき、マージしたドキュメントにおけるドキュメントD(T1)上のノード集合N1と同じ部分のデータがトランザクションT2によって更新されているので、ノード集合N1とノード集合N2とは異なることになる。ここで、異なるドキュメントD(T1)上のノードとD(T2)上のノードとが等価であるということは、そのノードがドキュメントD-stの同じノードからコピーされていることを意味する。例えば、トランザクションT2が削除したデータと同じ部分のデータをトランザクションT1のREAD(path)が参照する場合、参照されるノード集合N1にあるノードがノード集合N2の中には存在しない。ノード集合N1とノード集合N2とが同じであるとは、そのすべての要素がお互いに等価なノードであるということの意味し、ノード集合N1とノード集合N2とは等価であるということとする。

【0087】

RWアクセス衝突の検査は、ドキュメントD(T1)に対してREAD(path)がpathの評価時に参照するノード集合と、ドキュメントD(T1)とドキュメントD(T2)とをマージしたドキュメントに対してREAD(path)がpathの評価時に参照するノード集合とが等価であるか否かの検査に等しい。

【0088】

次に、異なる2つのドキュメントに対して読出しアクセスREAD(path)がpathの評価時に参照するノード集合の等価性検査について詳しく説明する。

【0089】

例えば、図3のドキュメントに対してREAD("flower/color")を行うとき、まず、名前が"flower"である子ノード{ノードn1,

ノードn2, ノードn3} (=ノード集合R1とする) が探索され、次に、それらのノードを出発点 (XPathの仕様では「コンテキストノード」と呼ばれる) として子ノードで名前が“color”である {ノードn5, ノードn8, ノードn11} (=ノード集合R2とする) が探索される。この場合、ノード集合R1→ノード集合R2の経路をたどって、結果のノード集合R2が特定されるので、pathの評価においてノード集合R1とノード集合R2との両方が参照される。したがって、異なるドキュメントに対して読出しアクセスが参照するノード集合が等価であるかを検査するためには、pathの探索経路上の各ステップにおいて参照されるそれぞれのドキュメント上のノード集合を比較して等価であるか検査すればよい。

【0090】

RWアクセス衝突検査において2つのドキュメントに対して読出しアクセスの参照するノード集合が等価であるということは、読出しアクセスがpathの評価時に参照するすべてのノード集合、言い換えれば、pathの探索経路上のすべてのステップで参照されるノード集合が等価であるということである。

【0091】

ところで、すべてのステップでのノード集合を比較せずに、RWアクセス衝突検査を効率的に行うように実施することもできる。以下、その方法について説明する。

【0092】

各ドキュメントにおいて、ツリー上で親ノードが書込みアクセスで更新されていれば、その子ノードも更新されていることになる。ドキュメントに対する書込みアクセスの操作には、大きく3つの操作、すなわち、INSERT (挿入)、DELETE (削除)、REPLACE (値の変更) の操作がある。例えば、トランザクションT2がドキュメントD (T2) に対して挿入の書込み操作を行ったとすると、ドキュメントD (T2) のドキュメントツリーにおいて挿入されたノードを根とする部分木にあるすべてのノードも、トランザクションT2によって新たに挿入されたノードである。また、トランザクションT2が削除の操作を行ったとすると、削除されたノードおよびそのノードを根とする部分木は、ドク

メントD (T2) のドキュメントツリー上には存在しない。また、データの値はドキュメントツリーの葉ノード (リーフノード) に格納されているので、値を更新するREPLACE操作は、葉ノードに対してのみ行われる (もしツリーの葉ではないノードの名前を更新するといったREPLACE操作を考える場合でも、そのノードを根とする部分木も変更されるものと仮定することで同様に考えることができる)。

【0093】

このように、1つのドキュメントツリー上で親ノードが書込みアクセスで更新されていればその子ノードも更新されているので、pathの探索経路上のあるステップで参照したノード集合が異なれば、次のステップでそれらのノード集合を出発点としてその部分木をたどって探索したノード集合も異なる。

【0094】

このことから、各ステップがツリーにおいて下向きの探索を続けるかぎり、途中のステップでは、参照したノード集合の比較をして等価性を検査する必要はない。

【0095】

ただし、XPathには、指定した条件を満たす子ノードや子孫ノードなどを探す下向きの探索の他に、親ノード、兄弟ノードなどを探す異なる方向の探索がある。そのように探索の方向が変わる前のステップでは、参照したノード集合の比較を行い等価であるかどうかを検査すればよい。例えば、図3のドキュメントに対してREAD (“flower [name=Tulip] /color”) を行うとき、結果に至る探索経路は {ノードn4} (=ノード集合R11) → {ノードn1} (=ノード集合R12) → {ノードn5} (=ノード集合R13) であり、ノード集合R11からノード集合R12への探索は下向きではないので、ノード集合R11におけるノードの比較は必要であるが、ノード集合R12からノード集合R13への探索は下向きであるので、ノード集合R12におけるノードの比較は省略できる (ノード集合R12での比較でノードが異なればノード集合R13での比較でもノードは異なるので、ノード集合R13の比較だけで十分である)。この場合は、ノード集合R11とノード集合R13を参照したステ

ップでノード集合の等価性を検査すればよい。

【0096】

XPath式の評価において探索の方向が変わることから、途中のステップで参照されるノード集合の等価性検査を行わなければならない場合は、大きく3つに分けられる。

【0097】

1つ目は、1つのXPath式の中に複数のパスが存在する場合である。この場合には、それぞれのパスを評価して得られたノード集合の等価性を検査する。例えば、XPathの仕様には、+、-などを含む様々な演算子や関数などがあり、 $path = path_1 + path_2$ のような例で $path$ の中に2つのパス $path_1$ と $path_2$ が存在する。したがって、 $path_1$ で参照するノード集合と $path_2$ で参照するノード集合とのそれぞれに対して等価性調査を行う。

【0098】

2つ目は、前述したように、1つのパスの中でも下向きではない探索方向に変わる場合である。XPathでは探索の方向について、軸(*axis*)を指定することによって設定でき、例えば、コンテキストノードに対する親ノード(*parent*)、子孫ノード(*ancestor*)などを探索するように設定することができる。その他に下向きではない方向の探索の軸としては、前の兄弟ノード(*preceding-sibling*)、後の兄弟ノード(*following-sibling*)などがある。

【0099】

3つ目は、XPathが定めたノードの位置情報によって探索を行う場合である。例えば、 $path = flower[position() = 2]$ の例のように、2番目の $flower$ ノードを探索する場合は、その位置に影響を及ぼす1番目のノードも参照の対象としてノード集合の等価性検査を行う。

【0100】

以上のように、トランザクションT1の読出しアクセスがトランザクションT2の書込みアクセスに対して起こすRWアクセス衝突は、ドキュメントD(T1)と、ドキュメントD(T1)及びドキュメントD(T2)をマージしたドキュ

メントとに対してREAD (path) を行って参照されるそれぞれのノード集合が等価であるか否かを検査することによって検出する。

【0101】

さて、トランザクションの分離性を保証するためには、並列に処理中の他のすべてのトランザクションに対して、要求された読出しアクセスがRWアクセス衝突を起こさないかを検査する必要がある。例えば、トランザクションT1が読出しアクセスを要求したときは、トランザクションT1以外の処理中のすべてのトランザクションに対してRWアクセス衝突の検査を行えばよい。これには、トランザクションT1とそれ以外の並列に処理中の1つのトランザクションとのRWアクセス衝突の検査を、トランザクションT1以外の並列に処理中のすべてのトランザクションを対象として繰り返し行う方法や、ドキュメントD(T1)に対して読出しアクセスが参照するノード集合と、ドキュメントD(T1)及び他のトランザクション用のすべてのドキュメントDをマージしたドキュメントとに対して読出しアクセスが参照するノード集合の等価性検査を行う方法がある。

【0102】

本実施形態では、ドキュメントD-allを用いることによってRWアクセス衝突の検査をさらに効率よく処理できる。すなわち、すべてのトランザクションが行ったデータの更新結果は1つのドキュメントD-all上に反映されている。したがって、並列に処理中の他の各トランザクションごとのドキュメントDを使用せずに、ドキュメントD(T1)に対して読出しアクセスが参照するノードの集合とドキュメントD-allに対して読出しアクセスが参照するノードの集合とが等価であるかを調べる1回の操作だけで、必要なRWアクセス衝突の検査が実施できる。

【0103】

(WRアクセス衝突の検査)

次に、WRアクセス衝突の検査について説明する。

【0104】

WRアクセス衝突の検査もRWアクセス衝突の検査と同様な考え方でノードの集合とノードの集合の比較によって行う。

【0105】

WRアクセス衝突は、あるトランザクションT1が要求する書込みアクセスが、他のトランザクションT2がすでに行った読出しアクセスに対して衝突を起こすことである。この衝突は、トランザクションT2がすでに行ったある読出しアクセスで参照したデータと同じ部分のデータに対してトランザクションT1が書込みアクセスを要求する場合に起きる。トランザクションT1が要求した書込みアクセスの操作をWとする。Wは、INSERT、DELETE、REPLACEのいずれかの操作である。

【0106】

まず、以前にトランザクションT2がある読出しアクセスREAD (path) を行った時点のドキュメントD (T2) の状態をD' (T2) として、ドキュメントD' (T2) に対してREAD (path) がpathの評価時に参照したノードの集合をN11とする。

【0107】

次に、書込みアクセスWを含むトランザクションT1のこれまでの更新結果をドキュメントD' (T2) へ反映させたドキュメントをD'' (T2) として、ドキュメントD'' (T2) に対して同じ読出しアクセスREAD (path) を行ったときに参照するノードの集合をN12とする。

【0108】

トランザクションT1が要求する書込みアクセスWとトランザクションT2が以前に行った読出しアクセスREAD (path) が衝突してWRアクセス衝突が起きるとき、ドキュメントD'' (T2) においてトランザクションT2のREAD (path) で参照されるデータと同じ部分のデータがトランザクションT1のWによって更新されているので、2つのドキュメントD' (T2) とD'' (T2) に対して読出しアクセスで参照したノードの集合N11とN12とは異なる。

【0109】

したがって、WRアクセス衝突の検査は、ドキュメントD' (T2) に対するREAD (path) の参照するノード集合と、ドキュメントD'' (T2) に

対する READ (p a t h) の参照するノード集合とが等価であるかの検査に等しい。

【0110】

トランザクションの分離性を保証するためには、並列して処理中の他のトランザクションがすでに行ったすべての読出しアクセスに対して、要求された書込みアクセスが WR アクセス衝突を起こさないかを検査する。例えば、トランザクション T1 が書込みアクセスを要求したときは、トランザクションリスト 123 にあるトランザクション T1 以外の各トランザクション T2 が行ったすべての読出しアクセスに対して WR アクセス衝突の検査を行う。

【0111】

まず、各読出しアクセスが行われた時点のドキュメント D' (T2) は、ドキュメント D-s t に対してその読出しアクセスの前に行われたすべての書込みアクセスの更新結果を反映したドキュメントであるので、ドキュメント D-s t に対してトランザクション T2 のトランザクションアクセス系列の書込みアクセスを行う方法で再作成できる。各読出しアクセスの READ (p a t h) が参照したノード集合 N11 は、再作成したドキュメント D' (T2) に対して READ (p a t h) が参照するノード集合を求めることによって得られる。

【0112】

なお、他の方法として、すべての読出しアクセスに対して参照したノード集合 N11 を保存しておくようにしてもよい。

【0113】

次に、書込みアクセス W の更新を反映したドキュメント D (T1) の状態を D' (T1) とすると、W を含むトランザクション T1 のこれまでの更新結果をドキュメント D' (T2) に反映したドキュメント D' (T2) は、ドキュメント D' (T1) と D' (T2) をマージすることによって得られる。

【0114】

なお、他の方法として、ドキュメント D' (T1) に対してトランザクション T2 のトランザクションアクセス系列の書込みアクセスを行うことによってドキュメント D' (T2) を再作成するようにしてもよい。

【0115】

図7の第1の構成例のリソースマネージャ12では、WRアクセス衝突の検査のときに、ドキュメントD-s tに対してトランザクションアクセス系列124をトレースしながら、すなわち、トランザクションのトランザクションアクセス系列124の読出しアクセスおよび書き込みアクセスを順番に実行させていきながら、すでに行われた読出しアクセス時のドキュメントDの状態を再作成し、読出しアクセスのREAD (p a t h) が参照するノード集合の等価性判定を行う。

【0116】

なお、他の方法として、トランザクションの書き込みアクセスを行ってドキュメントDを更新するときに、更新前のドキュメントDの状態を記録しておくように実施することも可能である。この場合は、ドキュメントDの再作成を行わなくてもよいが、ドキュメントDの更新毎にその状態を記録すると使用する記録容量が大きくなるというトレードオフがある。後で説明する第2の構成例のリソースマネージャ12では、ドキュメントDの状態を記録するタイミングをスケジュールしながらWRアクセス衝突の検査を行う。

【0117】

以下では、本構成例のリソースマネージャ12が行う処理手順について、(1) トランザクションの処理を開始するときの処理手順、(2) トランザクションが読出しアクセスを要求したときの処理手順、(3) トランザクションが書き込みアクセスを要求したときの処理手順、(4) トランザクションを中断後に再開するときの処理手順、(5) トランザクションをコミットするときの処理手順、(6) トランザクションをアボートするときの処理手順の順番に説明する。

【0118】

(1) トランザクションの処理を開始するときの処理手順

図11に、トランザクション識別子T i dのトランザクションの処理を開始するときの処理手順例を示す。

【0119】

まず、トランザクション識別子T i dをトランザクションリスト123に追加

する（ステップS1）。また、新しいトランザクションのために、トランザクションアクセス系列AS（Tid）とドキュメントD（Tid）を作成する（ステップS2，S3）。

【0120】

なお、トランザクションアクセス系列の初期値は、空リストである。

【0121】

また、ドキュメントD（Tid）は、ドキュメントD-stをコピーして作成する。なお、コピーの際には、例えば、ドキュメントD（Tid）の各ノードからドキュメントD-stの対応する各ノードへポインタをつけるなどの方法を実施すれば、アクセス衝突の検査において等価なノードであるか否かの比較を容易に行うことができる。

【0122】

以降のトランザクションTidのアクセスは、ドキュメントD（Tid）に対して行われる。

【0123】

（2）トランザクションが読出しアクセスを要求したときの処理手順

図12に、トランザクション識別子Tidのトランザクションが読出しアクセスREAD（path）を要求したときの処理手順例を示す。

【0124】

Eval（ドキュメント名1，ドキュメント名2，読出しアクセス）は、ドキュメント名1で指定したドキュメントとドキュメント名2で指定したドキュメントに対して読出しアクセスREAD（path）のpathを評価して結果のノード集合を返す関数を表す。pathの評価時には、探索の途中で必要に応じて参照するノード集合の等価性調査が行われ、もし等価でなければ探索を中断してアクセス衝突を知らせる「conflict」という結果を返す。そうでなければ、最後まで探索を続け、結果のノード集合を返す。すなわち、Evalは、RWアクセス衝突の検査で説明した、ドキュメント名1のドキュメントとドキュメント名2のドキュメントに対して読出しアクセスが参照するノード集合の等価性比較を実施しながら読出しアクセスの結果を求める関数である。

【0125】

まず、Eval (D (Tid), D-all, READ (path)) の結果を求める (ステップ S11)。結果が「conflict」であれば、RW アクセス衝突が起きる。そうでなければ、RW アクセス衝突は起きない。

【0126】

RW アクセス衝突がない場合は (ステップ S12)、読出しアクセスの結果を、トランザクションマネージャ 11 を介してアプリケーションプログラム 5 に返して、処理を続ける (ステップ S13)。また、トランザクションアクセス系列 AS (Tid) に READ (path) を記録する (ステップ S14)。

【0127】

RW アクセス衝突がある場合は (ステップ S12)、読出しアクセスがどのトランザクションの書込みアクセスと衝突するかを調べて、そのトランザクションが終了するまで待たなければならない。調査では、トランザクションリスト 123 の中から Eval (D (Tid), D (Tid'), READ (path)) = conflict であるトランザクション識別子 Tid' を見つける (ステップ S15)。そして、読出しアクセス処理を中断してトランザクション待ちグラフ 122 に (Tid' → Tid) を加える (ステップ S16)。トランザクション Tid は、トランザクション Tid' の処理が終了まで待機する。

【0128】

図 13 に、関数 Eval の処理手順例を示す。

【0129】

まず、ドキュメント D1 に対して path の最初のステップ s の評価を始めるとともに、ドキュメント D2 に対して path の最初のステップ s の評価を始める (ステップ S21)。

【0130】

ドキュメント D1 に対してステップ s の評価時に参照されたノード集合を N1 とし、ドキュメント D2 に対してステップ s の評価時に参照されたノード集合を N2 とする (ステップ S22)。

【0131】

ここで、ノード集合N1とノード集合N2とが等価でない場合（ステップS23）、 $Eval(D1, D2, READ(path)) = conflict$ を返して終了する（ステップS24）。

【0132】

ノード集合N1とノード集合N2とが等価である場合（ステップS23）、sがpathの最後のステップでなければ（ステップS25）、 $s = path$ の次のステップとして（ステップS26）、ステップS22から繰り返し、sがpathの最後のステップであれば（ステップS25）、 $Eval(D1, D2, READ(path)) = \text{結果のノード集合}$ を返して終了する（ステップS27）。

【0133】

（3）トランザクションが書込みアクセスを要求したときの処理手順

図14に、トランザクション識別子Tidのトランザクションが書込みアクセスを要求したときの処理手順例を示す。

【0134】

MERGE（ドキュメント名1，ドキュメント名2）は、ドキュメント名1で指定したドキュメントとドキュメント名2で指定したドキュメントをマージした結果のドキュメントを返す関数を表す。

【0135】

GetDoc（ドキュメント名，書込みアクセス）は、ドキュメント名で指定したドキュメントに対して書込みアクセスで指定した操作の更新結果を反映したドキュメント返す関数を表す。

【0136】

まず、WRアクセス衝突の検査のために、 $D(Tid)$ に要求された書込みアクセスWを行って、更新結果を反映したドキュメント $D-cand = GetDoc(D(Tid), W)$ を求める（ステップS31）。Wは、INSERT(node, data)、INSERT(node, child-data)、DELETE(node)、REPLACE(node, data)のいずれかの操作を指す。また、TL＝トランザクションリスト-Tid-アクセス系列が空で

あるトランザクション識別子とする（ステップS32）。

【0137】

そして、トランザクションリストにある他のトランザクションの中でトランザクションアクセス系列が空ではない個々のトランザクションに対してステップS34～S40で示す処理を行う。

【0138】

TL=NULLでなければ（ステップS32）、TLの中の最初のトランザクションのトランザクション識別子をxidとして、トランザクションxidのために、ドキュメントD-stをコピーしてドキュメントDocを用意し、トランザクションアクセス系列AS(xid)の最初のアクセス記録を取り出して、これをaccessとする（ステップS34）。

【0139】

accessが読出しアクセスであれば（ステップS35）、R=accessの操作READ(path)とし、D'=MERGE(Doc, D-cand)として、Eval(D', Doc, R)を求める（ステップS36）。

【0140】

Eval(D', Doc, R)の結果がconflictであれば（ステップS37）、WRアクセス衝突があるので、書込みアクセスの処理を中断してトランザクション待ちグラフ122に(xid→Tid)を加えて処理を終了する（ステップS38）。トランザクションTidは、トランザクションxidの処理が終了するまで待機する。

【0141】

Eval(D', Doc, R)の結果がconflictでなければ（ステップS37）、WRアクセス衝突はないので、ステップS40に移る。

【0142】

他方、ステップS35においてaccessアクセスが書込みアクセスであれば、W=accessの書込みアクセスの操作として、Doc=GetDoc(Doc, W)を実行して、ドキュメントDocに取り出した書込みアクセスWの更新を反映させ（ステップS39）、ステップS40に移る。

【0143】

access がトランザクションアクセス系列 AS (xid) の最後のアクセスでなければ (ステップ S40)、トランザクションアクセス系列 AS (xid) の次のアクセスを取り出し、これを access として (ステップ S41)、ステップ S35 に戻る。

【0144】

また、access がトランザクションアクセス系列 AS (xid) の最後のアクセスであれば (ステップ S40)、対応するトランザクションに対する衝突の検査は終了し、次いで、 $TL = TL - xid$ として (ステップ S42)、ステップ S32 に戻る。

【0145】

そして、ステップ S32 において $TL = NULL$ であれば、すなわち、対象となったすべてのトランザクションに対する WR アクセス衝突の検査をとおして衝突がなければ、 $D(tid) = D - cand$ 、 $D - all = GetDoc(D - all, W)$ として、書込みアクセス W の結果をドキュメント D (Tid) とドキュメント D - all の両方に反映させるとともに、書込みアクセス W をトランザクションアクセス系列 AS (Tid) に記録する (ステップ S33)。

【0146】

(4) トランザクションを中断後に再開するときの処理手順

トランザクションを中断後に再開するときには特別な処理は必要ない。中断していたアクセスが読出しアクセスである場合は、上記の (2) のトランザクションが読出しアクセスを要求したときの処理手順の最初に戻って処理を続ける。書込みアクセスである場合は、上記の (3) のトランザクションが書込みアクセスを要求したときの処理手順の最初に戻って処理を続ける。

【0147】

(5) トランザクションをコミットするときの処理手順

トランザクションをコミットして終了するときには、そのトランザクションが行ったデータの更新結果をファイルおよび他のトランザクションのドキュメントに反映させる処理 a と、そのトランザクションの終了を待ちながら中断している他

のトランザクションを再開させる処理 b とを行う。

【0 1 4 8】

トランザクション識別子 T i d のトランザクションをコミットするときは、まず処理 a のために、ドキュメント D (T i d) をその時点のファイルのドキュメント D - s t にマージして、ハードディスク 3 上のファイル 3 1 に記録する。また、ドキュメント D (T i d) をトランザクションリスト 1 2 3 に記録されている各トランザクションに対応するドキュメント D にマージする。この操作によって、中断しているものを含むすべてトランザクションのドキュメント D にコミットされた更新結果を反映される。

【0 1 4 9】

ここでは、トランザクションのコミット時にコミットする更新結果を並行に処理している他のトランザクションへ知らせる場合を例にとって説明したが、この処理を省略あるいは後回しにするように実施することもできる。この処理を省略すると個々のトランザクションのドキュメント D には反映されていないが、すでにコミットしたデータの更新が存在することとなる。したがって、上記の (2) の処理手順において R W アクセス衝突を発見したときに、アクセス衝突の対象がすでにコミットしたトランザクションであれば、その時点でコミットした更新結果をアクセス衝突を起こしたトランザクションのドキュメント D に反映すればよい。

【0 1 5 0】

次に、処理 b のために、トランザクション待ちグラフ 1 2 2 からトランザクション識別子 T i d のトランザクションの終了を待っているトランザクションを見つけ出す。そのようなトランザクションがあれば、そのトランザクションの再開を指示する。また、待ちグラフからトランザクション T i d を表す点とその点を始点とする辺をすべて削除する。

【0 1 5 1】

処理 a と処理 b が終了すれば、最後に、トランザクション識別子 T i d をトランザクションリスト 1 2 3 と待ちグラフから削除し、トランザクションアクセス系列 A S (T i d) とドキュメント D (T i d) も削除する。

【0152】

(6) トランザクションをアボートするときの処理手順

トランザクションをアボートして終了するときには、そのトランザクションが行ったデータの更新結果を破棄する。ドキュメントD-a11には処理中のすべてのトランザクションが行った更新結果が反映されているので、アボートするトランザクションが行った更新結果も反映されている。それを破棄するために、ドキュメントD-a11を再作成する処理を行う。また、コミット時と同様に、そのトランザクションの終了を待ちながら待機している他のトランザクションを再開させる処理を行う。

【0153】

トランザクション識別子Tidのトランザクションをアボートするときには、まず、トランザクション識別子Tidをトランザクションリスト123から削除し、トランザクションアクセス系列AS(Tid)とドキュメントD(Tid)も削除する。

【0154】

次に、待ちグラフからトランザクション識別子Tidのトランザクションの終了を待っているトランザクションを見つけ出す。そのようなトランザクションがあれば、そのトランザクションの再開を指示する。また、待ちグラフからトランザクションTidを表す点とその点を始点とするすべての辺を削除する。

【0155】

最後に、その時点のファイルのドキュメントD-stにトランザクションリスト123にあるすべてのトランザクションのドキュメントDを重ねてマージさせることで、ドキュメントD-a11を再作成する。この処理によってドキュメントD-a11は、アボートするトランザクションを除くすべての処理中のトランザクションの更新結果を反映していることとなる。

【0156】

(リソースマネージャの第2の構成例)

次に、リソースマネージャの第2の構成例について説明する。

【0157】

第2の構成例が第1の構成例と異なる点は、WRアクセス衝突の検査の方法である。第1の構成例では、リソースマネージャ12は、トランザクションのトランザクションアクセス系列をトレースして以前の読出しアクセス時のドキュメントDの状態を再作成しながらWRアクセス衝突の検査を行う。第2の構成例では、リソースマネージャ12は、各ドキュメントDの以前の状態を再作成する方法に代わって、ドキュメントD-a11の以前の状態を用いる方法によってWRアクセス衝突の検査を行う。

【0158】

(WRアクセス衝突の検査)

以下、WRアクセス衝突の検査について説明する。

【0159】

図15は、あるトランザクションTidのトランザクションアクセス系列の一例を示している。Tidは、トランザクション識別子である。トランザクションは、読出しアクセスと書き込みアクセスからなるアクセス系列を持つ。図15において、縦線は連続した読出しアクセスの系列(1つの読出しアクセスのみからなる系列である場合を含む)を表し、四角は書き込みアクセスを表し、上下に連続した四角は連続した書き込みアクセス系列を表す。以降、トランザクションTidの連続した読出しアクセスの系列をRSTidと表記し、連続した書き込みアクセス系列WSTidと表記する。また、WSTid(i)はトランザクション処理の開始以降のi番目のWSTidを表し、RSTid(i)はWSTid(i)の後に続くRSTidを表す。最初書き込みアクセス以前の読出しアクセスの系列は、RSTid(0)とする。

【0160】

ここでは、リソースマネージャ12が図16に例示するようなトランザクションアクセス系列を持つ3つのトランザクションT1、トランザクションT2、トランザクションT3を処理しているときに、Time6の時点でトランザクションT1が書き込みアクセスWを要求した場合のWRアクセス衝突の検査を例にとって説明する。

【0161】

リソースマネージャ12は、この書込みアクセスWが、並列に処理中の他のトランザクション、すなわち、トランザクションT2とトランザクションT3とがすでに行った読み込みアクセスと衝突しないかを検査する必要がある。すなわち、トランザクション2の $RS_{T2}(0)$ と $RS_{T2}(1)$ と $RS_{T2}(2)$ のすべての読出しアクセスおよびトランザクションT3の $RS_{T3}(0)$ と $RS_{T3}(1)$ と $RS_{T3}(2)$ のすべての読出しアクセスが、WRアクセス衝突検査の対象である。

【0162】

トランザクションT1がドキュメントD(1)に対して書込みアクセスWを行った更新後のドキュメントをD'(1)とする。

【0163】

第1の構成例で説明したように、例えば、 $RS_{T2}(1)$ の読出しアクセスRとWRアクセス衝突を検査するときは、Time1の時点のドキュメントD(2)と、ドキュメントD(2)およびドキュメントD'(1)をマージしたドキュメントに対する読出しアクセスRの参照するノード集合を比較する。

【0164】

すべての読出しアクセスに対してこの検査を行うので、第1の構成例では、トランザクションT1とトランザクションT2のトランザクションアクセス系列を順番に実行してTime1とTime5の時点のD(2)と、Time3とTime4の時点のD(3)を求める。

【0165】

これに対して、第2の構成例では、ドキュメントD-allを用いてWRアクセス衝突を検査する。Time1の時点のドキュメントD(2)は、ドキュメントD-stに対して $WS_{T2}(1)$ の書込みアクセスの更新結果を反映したものである。この更新はTime1の時点のD-allにも反映されているので、Time1の時点のD(2)の代わりに、Time1の時点のD-allに対して読出しアクセスRを行っても、その結果は同じである。したがって、アクセス衝突の検査は、Time1の時点のD-allに対する読出しアクセスRの参照するノード集合と、D'(1)とTime1の時点のD-allとをマージしたドキュメントに対する読出しアクセスRの参照するノード集合とが同じであるかを比

較することと等価になる。同様に、 $RS_{T2}(0)$ と $RS_{T3}(0)$ に対する検査では初期時点の $D-a11$ すなわちドキュメント $D-st$ を、 $RS_{T2}(2)$ に対する検査では $Time 5$ の時点の $D-a11$ 、 $RS_{T3}(1)$ 、 $RS_{T3}(2)$ に対する検査では $Time 3, 4$ の時点の $D-a11$ を使用すればよい。第2の構成例では、更新されるドキュメント $D-a11$ の各状態を記録しておいて、以降の WR アクセス衝突の検査で使用する。

【0166】

ただし、記録容量が十分確保できる場合には、すべての時点において $D-a11$ の状態を記録しておくができるが、他方、記録容量が限られている場合には、すべての時点において $D-a11$ の状態を記録しておくことはできないため、どの時点の $D-a11$ を記録するのが効果的であるかを決定しなければならない。

【0167】

例えば、 $RS_{T2}(1)$ の読出しアクセスに対する衝突の検査を行うときは、 $Time 1$ の時点の $D-a11$ を使用しても、 $Time 3, 4$ の時点の $D-a11$ を使用してもよい。なぜなら、トランザクション $T2$ の $WS_{T2}(1)$ の更新が $D-a11$ に反映された $Time 1$ 以降から、トランザクション $T2$ の $WS_{T2}(2)$ が次の更新 T を $D-a11$ に反映する $Time 5$ の前であれば、どの時点の $D-a11$ に対する読出しアクセス R の参照するノード集合も等価であるからである。ドキュメント $D-a11$ にはリソースマネージャ 12 が並行に処理しているすべてのトランザクションの更新結果が反映されているが、それらの更新はお互いにアクセス衝突を起こさないものである。

【0168】

もし $Time 1$ の時点の $D-a11$ に対する読出しアクセス R の参照するノード集合と、 $Time 2$ の時点の $D-a11$ に対する読出しアクセス R の参照するノード集合とが異なれば、 $Time 2$ の時点での $D-a11$ を更新したトランザクション $T1$ の書込みアクセスが R と衝突するということである（ただし、 $Time 5$ での $D-a11$ の更新は、他のトランザクションではなくトランザクション $T2$ の書込みアクセスによって行われるので、 $Time 5$ での $D-a11$ に対するトランザクション $T2$ の読出しアクセス R の参照するノード集合は、それ以

前の結果と同じではない)。

【0169】

このような理由により、この例では、 $RS_{T2}(1)$ と $RS_{T3}(1)$ に対するWRアクセス衝突の検査で同じTime 3の時点のD-a11を使用できる。第2の構成例では、Time 3時点のD-a11のように複数のRSに対するWRアクセス衝突の検査で利用できるD-a11を選択して記録する。どのタイミングでD-a11の状態を記録するかを決める方法については以降で詳しく説明する。

【0170】

図17に、第2の構成例に係るリソースマネージャの構成例を示す。図中、12はリソースマネージャ、121はドキュメントD-a11、122はトランザクション待ちグラフ、123はトランザクションリスト、124はトランザクションアクセス系列、125はドキュメントD(Tid)、126は記録数を、127はドキュメントD-sを、128はS-Point管理表を、3はハードディスクを、31はドキュメントD-st(図1のファイル31)をそれぞれ示す。

【0171】

以下では、第1の構成例のリソースマネージャ12と相違する点を中心に説明する。

【0172】

図17のトランザクションアクセス系列124は、第1の構成例のリソースマネージャ12と同様に、個々のトランザクションが処理の開始から行ってきた読出しアクセスおよび書込みアクセスの系列をリストとして記録管理している。ただし、読出しアクセス系列と書込みアクセス系列に加えて、書込みアクセス系列の書込みアクセス数を管理する。以降で説明するが、書込みアクセス数は、どの時点でD-a11の状態を記録するかを決定するために用いられる。

【0173】

図18に、トランザクションアクセス系列の一構成例を示す。この例は、図15で例示したトランザクションTidのトランザクションアクセス系列と同じ例

である。

【0174】

トランザクションアクセス系列AS (Tid) は、読出しアクセス系列と書込みアクセス系列のリストであり、RS (i) とWS (i) には、それぞれ、トランザクションTidのi番目の読出しリストRSTid (i) の読出しアクセス操作のリストと、i番目の書込みアクセスリストWSTid (i) の書込みアクセス操作のリストが記録されている。

【0175】

図17の記録数H (図中の126) は、ドキュメントD-a11の更新前の状態を何個まで記録できるかを示す数値である。記録数Hが大きければ大きいほどD-a11の状態を多く記録できるのでWRアクセス衝突検査の効率が上がる。反面、D-a11の記録に必要な記憶容量が大きくなるというトレードオフがある。記録数Hは、トランザクション処理システムが初期に設定するが、トランザクションの処理中にその値を変更してもよい。

【0176】

図17のドキュメントD-s (図中の127) は、過去のある時点のドキュメントD-a11の状態を記録している。以降、D-a11を記録した時点をもS-Pointと呼び、ある時点でD-a11を記録すると決定することをS-Pointを設定すると呼ぶ。リソースマネージャ12は、記録数H個までのS-Pointを設定できるので、H個までのD-sを記録管理している。i番目に設定されたS-Pointで記録されたD-sをD-s (i) と表記する。

【0177】

図17のS-Point管理表 (図中の128) には、記録数H個までのエントリがあり、各エントリは個々のS-Pointに対応する。各エントリには、対応するS-Pointを設定した時点において各トランザクションが何番目の書込みアクセス系列WSまでの更新結果をD-a11に反映しているかを示す情報と、そのS-Pointの設定によって得られる効果の大きさを示す情報を記録管理している。

【0178】

以下、リソースマネージャ 12 が S-P o i n t 管理表 128 に記録している情報を利用して S-P o i n t の設定を決定する方法について説明する。

【0179】

図 19 は、図 16 と同じトランザクション T1、トランザクション T2、トランザクション T3 のトランザクションアクセス系列を表している。ただし、T i m e 1 から T i m e 5 の各時点が図 16 とは異なる例である。

【0180】

トランザクションが要求した各書込みアクセスが分離性を破らないと判断されて D-a l l を更新することになったときに、S-P o i n t を設定するか否か、すなわち、その時点の D-a l l の状態を D-s として記録しておくか否かを決定する。

【0181】

図 20 は、図 19 の T i m e 1 の時点で 1 つ目の S-P o i n t が設定され、T i m e 2 の時点で 2 つ目の S-P o i n t が設定されたときの S-P o i n t 管理表 128 を示す。

【0182】

S-P o i n t 管理表 128 の各エントリーは、1 つの S-P o i n t に対応していて、各エントリーの S-P o i n t 番号は、対応する S-P o i n t が何番目の S-P o i n t であるかを示す。

【0183】

S-P o i n t エントリーには、まず、リソースマネージャ 12 が処理中の各トランザクションに対応したそれぞれの WS 番号の欄がある。各トランザクションの WS 番号の欄には、S-P o i n t が設定された時点においてそのトランザクションの最近の書込みアクセスリストが何番目の WS であったかを記録している。S-P o i n t エントリーの各トランザクションの WS 番号から、S-P o i n t の時点で記録した D-a l l (すなわち、D-s) に、そのトランザクションの何番目の WS までの更新結果が反映されているかがわかる。したがって、S-P o i n t に対応する D-s を利用して、そのトランザクションの WS 番目の読出しアクセス系列の読出しアクセスに対する WR アクセス衝突を検査できる

ということがわかる。

【0184】

例えば、図19において、1番目のS-P o i n tが設定されたT i m e 1の時点におけるトランザクションT2の最近の書込みアクセスリストW S_{T2} (1)は、1番目のWSである。よって、図20において、S-P o i n t番号が1のエントリーにおけるトランザクション2のWS番号は1である。他のトランザクションT1とT3に対しては、最近の書込みアクセスリストがないので、0となっている。同様に、2番目のS-P o i n tが設定されたT i m e 2の時点におけるトランザクションT1の最近の書込みアクセスリストW S_{T1} (1)は、1番目のWSであるので、図20のS-P o i n t番号が2のエントリーのトランザクション1のWS番号は1となっている。

【0185】

S-P o i n tエントリーには、次に、対応するS-P o i n tを設定することによって得られる効果の大きさを表す効果値の欄がある。S-P o i n tを設定してD-a l lの状態をD-sとして記録しておく、以降のWRアクセス衝突の検査においてD-sを利用することができるので、D-a l lの状態を再現するために必要なコストが削減できる。S-P o i n t設定以降のWRアクセス衝突の検査のたびにそのコストを削減できるので、S-P o i n tの設定によって得られる効果の大きさは、S-P o i n tを設定しなかったときにその時点のD-a l lを再現するために必要なコストに比例する。S-P o i n tの時点のD-a l lを再現するためには、各トランザクションに対するその時点において最近の書込みアクセスリスト（すなわち、WS番目の書込みアクセスリスト）の書き込みアクセスをもう一度行い、D-a l lに反映された更新を再現する必要がある。

【0186】

したがって、S-P o i n tの効果値は、S-P o i n tエントリーにおける各トランザクションのWS番目の書込みアクセスリストの書込みアクセス数の和とする。

【0187】

ただし、S-P o i n t エントリーにおいてトランザクションのWS番号が1つ前のS-P o i n t エントリーにおけるWS番号と同じである場合は、1つ前のS-P o i n t のD-sにWS番目の書き込みアクセスリストの更新結果がすでに反映されているので、そのトランザクションのWS番目の書き込みアクセスリストの書き込みアクセス数は効果値に足さない。例えば、図20の1番目のS-P o i n t の効果値は、WS_{T2}(1)の書き込みアクセス数から1となっている。2番目のS-P o i n t の効果値は、WS_{T1}(1)の書き込みアクセス数から3となっている。2番目のS-P o i n t のエントリーのトランザクション2のWS番号欄が1であるが、1つ前の1番目のS-P o i n t エントリーのWS番号欄も同じく1であるのでWS_{T2}(1)の書き込みアクセス数は2番目のS-P o i n t の効果値に足さない。例えば、図19のTime 2で1番目のS-P o i n t が設定されたときは、S-P o i n t 管理表128は図21のようになり、1番目のS-P o i n t の効果値はWS_{T1}(1)の書き込みアクセス数にWS_{T2}(1)の書き込みアクセス数を足して3+1=4となる。

【0188】

リソースマネージャ12は、S-P o i n t 管理表128にある各トランザクションのWS番号を参照することでWRアクセス衝突の調査のときにどのD-sを利用できるかを知る。また、ある時点でS-P o i n t を設定するときの効果値を計算し、その値を基準として新しいS-P o i n t を設定するか否かを決定する。S-P o i n t 設定を決定する方法については、以降のトランザクションが書き込みアクセスを要求したときのリソースマネージャ12の処理手順の中で詳しく説明する。

【0189】

以下では、リソースマネージャ12が行う処理手順の一例について、(1) トランザクションの処理を開始するときの処理手順、(2) トランザクションが読み出しアクセスを要求したときの処理手順、(3) トランザクションが書き込みアクセスを要求したときの処理手順、(4) トランザクションを中断後に再開するときの処理手順、(5) トランザクションをコミットするときの処理手順、(6) トランザクションをアボートするときの処理手順の順番に説明する。

【0190】**(1) トランザクションの処理を開始するときの処理手順**

トランザクションの処理を開始するときの処理手順例は、図11で示した第1の構成例のリソースマネージャ12の処理手順例と同様である。

【0191】**(2) トランザクションが読出しアクセスを要求したときの処理手順**

トランザクションが読出しアクセスを要求したときの処理手順例は、図12で示した第1の構成例のリソースマネージャ12の処理手順例と同様である。ただし、図12のステップS14でREAD (path) をトランザクションアクセスリストAS (Tid) に追加するときは、もしAS (Tid) の最後のアクセスリストが読出しアクセスリストRS (i) であれば、そのリストの最後に追加記録し、書込みアクセスリストWS (i) であれば、新しいRS (i) を作成して、その最初のアクセスとして記録する。もしトランザクションの最初のアクセスであれば、RS (0) の最初のアクセスとして記録する。

【0192】**(3) トランザクションが書込みアクセスを要求したときの処理手順**

リソースマネージャ12は、まず、S-Point管理表128を調べて、利用可能なD-sを参照しながら、WRアクセス衝突の検査を行う。調査の結果、要求された書込みアクセスが衝突を起こさないことがわかれば、次に、その時点でS-Pointを設定するか否かを決定して、それから書込みアクセスの結果をD-allに反映する。

【0193】

以下では、S-Point管理表128のh番目のS-Pointエントリーの各トランザクションTidに対応するWS番号をMTid(h)と表記する。

【0194】

まず、WRアクセス衝突の検査について説明する。

【0195】

リソースマネージャ12は、トランザクションTidの要求した書込みアクセスWが、トランザクションリストにある他のすべてのトランザクションの読出し

アクセスリストに対して衝突を起こさないかを検査する必要がある。S-P o i n t 管理表 128 のエントリーの WS 番号欄に検査対象となる読出しアクセスリストの番号があれば、対応する S-P o i n t の D-s を利用する。番号がないときは、その時点の D-a l l を再作成する必要がある。ただし、各トランザクションの最初の読み込みアクセスリスト RS (0) に対して検査を行うときは、D-s t を利用することができ、最近の読み込みアクセスリストに対して検査を行うときは、D-a l l を利用することができる。現時点の D-a l l には、各トランザクションの最近の書き込み、すなわちトランザクションアクセスリストの最後の WS の書き込みの結果が反映されている。

【0196】

WR アクセス衝突の検査では、まず、トランザクション T i d の要求した書き込みアクセス W をドキュメント D (T i d) に反映したドキュメント D-c a n d を用意する。また、変数 h の初期値を最後の S-P o i n t の番号+1 とする。

【0197】

ここでは、S-P o i n t 管理表 128 の最後のエントリーから最初のエントリーまでを逆順に参照しながら衝突の検査を行う場合を例にとって説明するが、もちろん、どのような順番で行っても実施できる。

【0198】

変数 h が最後の S-P o i n t の番号+1 のときの処理は、各トランザクションの最近の読出しアクセスリスト RS に対する検査を示す。その時点の各トランザクションの最後の書き込みアクセスリストを WS (i) とすると、RS (i) を対象する検査である。すでに説明したように、この場合は、その時点の D-a l l を利用できるので、RS (i) の各読出しアクセス R に対して E v a l (D-a l l, M E R G E (D-a l l, D-c a n d), R) の結果が c o n f l i c t ではないかを調べる。すべてのトランザクションの各読出しアクセスに対して結果が c o n f l i c t でなければ、衝突は起きない。

【0199】

変数 h が 0 のときの処理は、各トランザクションの最初の読出しアクセスリスト RS (0) に対する検査を示す。すでに説明したように、この場合は、D-s

tを利用できるので、RS (i) の各読出しアクセスRに対してEval (D-st, MERGE (D-st, D-cand), R) の結果がconflictではないかを調べる。すべてのトランザクションの各読出しアクセスに対して結果がconflictでなければ、衝突は起きない。

【0200】

変数hがその他の値であるときは、各トランザクションに対して次の処理を行う。トランザクション識別子をxidとする。S-Point管理表128のh番目のエントリーからトランザクションxidのWS番号M_{xid}(h)を調べ、そのWS番号をiとする。なお、h番目のS-Pointを設定した時点で記録されたドキュメントD-s(h)にはWS(i)の更新結果が反映されているので、RS(i)に対するアクセス衝突検査でD-s(h)を利用できる。i=M_{xid}(h+1)であれば、RS(i)に対する検査はすでに行われているので検査の必要はない。そうでなければ、まず、RS(i)の各読出しアクセスRに対して、Eval (D-s(h), MERGE (D-s(h), D-cand), R) を調べる。次に、i=i+1としてRS(i)の次の読出しアクセスリストについて考える。もしi=M_{xid}(h+1)であれば、RS(i)に対する検査はすでに行われている。もしi<M_{xid}(h+1)であれば、RS(i)の読出しアクセスが行われた時点のD-allの状態は記録されていないので、再作成を行う必要がある。D-s(h)にWS(i)の更新操作を行って得られるドキュメントをDocとして、RS(i)に対する検査はDocを用いて行われる。すなわち、RS(i)の各読出しアクセスRに対してEval (Doc, MERGE (Doc, D-cand), R) の結果を調べる。この処理は、RS(i)の次の読出しアクセスリストが最後のRSである、あるいは、そのRSに対する検査がすでに行われている、という条件を満たすまで繰り返される。

【0201】

以上のようにして、すべてのトランザクションの読出しアクセスリストに対してWRアクセス衝突の検査が終了衝突がなければ、その時点でS-Pointを設定するか否かを定める処理に入る。

【0202】

その後に、書込みアクセスWの結果をドキュメントD (T i d) とドキュメントD-a l l の両方に反映させる。また、書込みアクセスWをトランザクションアクセス系列A S (T i d) に記録する。

【0203】

図22に、トランザクション識別子T i dのトランザクションが書込みアクセスWを要求したときのWRアクセス衝突の調査の処理手順の一例を示し。

【0204】

ステップS51で、D-c a n d=G e t.D o c (D (T i d) , W) 、h=最後のS-P o i n tの番号+1とする。

【0205】

ステップS52で、h=最後のS-P o i n tの番号+1ならば、ステップS53でD o c=D-a l lとし、h=0ならば、ステップS54でD o c=D-s tとし、それ以外ならば、ステップS55でD o c=M E R G E (D-s (h) , D-c a n d) とした後に、いずれの場合も、ステップS56で、T L=トランザクションリスト-T i d-アクセス系列が空でないトランザクション識別子とする。

【0206】

ステップS57で、T L=N U L Lの場合、ステップS58でh=0ならば、ステップS59に移って、この処理を終了し、次のS-P o i n t 決定フローチャート(図23参照)を実行する。他方、ステップS58でh=0でないならば、ステップS60で、h=h-1として、ステップS52に戻る。

【0207】

ステップS57で、T L=N U L Lでない場合、ステップS62で、R S=R x i d (i) 、R=R Sの最初のアクセス、D'=M E R G E (D O C, D-c a n d) とする。

【0208】

ステップS63で、E v a l (D' , D o c, R) =c o n f l i c tである場合、ステップS64で、待ちグラフに(x i d→T i d)を追加する。

【0209】

他方、ステップS63で、 $Eval(D', Doc, R) = conflict$ でない場合、ステップS65で、RがRSの最後のアクセスでなければ、ステップS66で、 $R = RS$ の次のアクセスとし、ステップS63に戻る。

【0210】

ステップS65で、RがRSの最後のアクセスであれば、ステップS67で、 $h = \text{最後の } S-Point \text{ の番号} + 1$ であるとき、あるいは、そうでなくても、ステップS68で、 $M_{xid}(h) = M_{xid}(h+1)$ であるとき、あるいは、そうでなくても、ステップS69で、 $i < M_{xid}(h+1)$ でないときは、ステップS70で、 $TL = TL - xid$ として、ステップS62に戻る。

【0211】

また。ステップS69で、 $i < M_{xid}(h+1)$ であるときは、ステップS71で、 $i = i + 1$ 、 $Doc = Doc$ にWS(i)の更新結果を反映したドキュメントとして、ステップS62に戻る。

【0212】

次に、S-Pointの設定について説明する。

【0213】

この処理は、書込みアクセスWをトランザクションアクセス系列の新しい書込みアクセスリストWS(i+1)の最初のアクセスとして記録する前に行う。

【0214】

まず、新しいS-Pointを設定したときに増える効果値を計算する。すでに説明したように、効果値は、すべてのトランザクションにおける最近の書込みアクセスリストWSの書込みアクセス数の和である。ただし、WSの番号が前に設定されたS-PointエントリーのWS番号と同じであれば、WSの書込みアクセス数は効果値に足さない。図23においては、変数eが、計算された効果値を表している。

【0215】

効果値(e)を計算した後、S-Point管理表128にある最後のS-Point番号を調べる。最後のS-Point番号は、その時点までに設定されたS-Pointの数である。その数を h' とする。

【0216】

h' が記録数 H より小さければ、新しい $S-P o i n t$ を設定する。 $S-P o i n t$ 管理表 128 に、新たに、 $h' + 1$ 番目の $S-P o i n t$ のエントリーを作成する。エントリーの $S-P o i n t$ 番号は $h' + 1$ であり、効果値は e である。各トランザクションに対応する $W S$ 番号には、トランザクションアクセス系列を調べて各トランザクションの最近の書込みアクセスリストの書き込みアクセス数を記録する。そして、その時点の $D-a l l$ を、 $D-s (h' + 1)$ として記録する。

【0217】

h' が記録数 H と同じであれば、それ以上の数の $S-P o i n t$ は設定できない。したがって、すでに設定した $S-P o i n t$ の中で取り消した場合に減る効果値が一番小さいものを調べ、新しい $S-P o i n t$ を設定して増える効果値と比較する。各 $S-P o i n t$ の取り消しによって減る効果値は、その $S-P o i n t$ を削除してもその次の $S-P o i n t$ (最近の $S-P o i n t$ の場合は、新しく設定する $S-P o i n t$) に移動するだけで消滅はしない値を効果値から引いたものである。例えば、ある h 番目の $S-P o i n t$ の効果値に、あるトランザクション $x i d$ の $W S$ 番目の書込みアクセスリストの書込みアクセス数 N が加算されているときについて考える。 $h + 1$ 番目の $S-P o i n t$ ($h = h'$ の場合は、新しい $S-P o i n t$) においてトランザクション $x i d$ が同じ $W S$ 番号である場合は、 h 番目の $S-P o i n t$ を取り消しても、 N は次の $h + 1$ 番目の $S-P o i n t$ (あるいは新しい $S-P o i n t$) の効果値に加算される。そうではない場合は、値 N 分の効果値は、 h 番目の $S-P o i n t$ の取り消しによって消滅する。

【0218】

取り消しによって減る効果値が一番小さかった $S-P o i n t$ の番号を $h-m i n$ とする。 $h-m i n$ 番目の $S-P o i n t$ を削除すると減る効果値と、新しい $S-P o i n t$ を設定すると増える効果値を計算して、後者が大きければ、 $h-m i n$ 番目の $S-P o i n t$ を取り消して新しい $S-P o i n t$ を設定する。 $h-m i n$ 番目の $S-P o i n t$ を削除すると減る効果値は、 $h-m i n$ 番目の

S-Pointの効果値から変数 e_1 の値を引いたものである。 e_1 の値は、各トランザクションに対して、 $h-min$ 番目のS-Pointに対応するWS番号 $M_{xid}(h-min)$ と、その次の $h-min+1$ 番目のS-Pointに対応するWS番号 $M_{xid}(h-min+1)$ が同じである場合に、WS番目の書き込みアクセスリストの書き込みアクセス数を足したものである。 $M_{xid}(h-min)$ と $M_{xid}(h-min+1)$ とが同じである場合は、 $h-min$ 番目のS-Pointを削除しても $h-min+1$ 番目のS-Pointのドキュメント $D-s(h+1)$ にトランザクション xid のWS番目の書き込みアクセスリストの更新結果が反映されているので、その分の効果値は減らずに、 $h-min+1$ 番目のS-Pointの効果値に足される。

【0219】

新しいS-Pointを設定すると増える効果値 e が $h-min$ 番目のS-Pointを削除すると減る効果値より大きければ、 $h-min$ 番目のS-Pointのエントリーと $D-s(h-min)$ を削除し、それ以降のS-Pointの番号と対応する $D-s$ の番号を1ずつ減らす。また、新しい $h-min$ 番目のS-Pointの効果値に e_1 を足す。そして、新しいS-Pointのエントリーを作成してその時点の $D-all$ を $D-s(H)$ として記録する。

【0220】

図23に、S-Point設定の処理手順の一例を示す。

【0221】

ステップS81で、 $h' =$ 最後のS-Pointの番号、 $TL =$ トランザクションリスト、 $xid = TL$ の中の最初のトランザクション識別子とする。

【0222】

ステップS82で、 $m = AS(xid)$ の最後のWSの番号とする。

【0223】

ステップS83で、 $m = M_{xid}(h')$ でないならば、ステップS84で、 $e = AS(xid)$ の最後のWSの書き込みアクセス数とし、他方、ステップS83で、 $m = M_{xid}(h')$ であるならば、ステップS84は、スキップする。

【0224】

ステップ S85 で、 $TL = NULL$ でない場合、ステップ S86 で、 $TL = TL -xid$ 、 $xid = TL$ の中の最初のトランザクション識別子として、ステップ S82 に戻る。

【0225】

ステップ S85 で、 $TL = NULL$ である場合、ステップ S87 で、 $h' < \text{記録数} H$ であるならば、ステップ S88 に移り、新しい $S-Point$ を設定し、その効果値 $= e$ として、この処理を終了する。

【0226】

他方、ステップ S87 で、 $h' < \text{記録数} H$ でないならば、ステップ S89 に移り、 $h-min = \text{削除によって減る効果が一番小さい } S-point \text{ の番号}$ 、 $TL = \text{トランザクションリスト}$ 、 $xid = TL$ の中の最初のトランザクション識別子とする。

【0227】

ステップ S90 で、 $e1 = 0$ とする。

【0228】

ステップ S91 で、 $h-min = h'$ である場合、ステップ S92 で、 $m = AS(xid)$ の最後の WS の番号とし、ステップ S93 で、 $m = M_{xid}(h-min)$ であるならば、ステップ S95 で、 $e1 = e1 + AS(xid)$ の $M_{xid}(h-min)$ 番目の書き込みアクセス数とし、 $m = M_{xid}(h-min)$ でないならば、ステップ S95 をスキップして、ステップ S96 に移る。

【0229】

他方、ステップ S91 で、 $h-min = h'$ でない場合、ステップ S94 で、 $M_{xid}(h-min) = M_{xid}(h-min+1)$ であるならば、ステップ S95 で、 $e1 = e1 + AS(xid)$ の $M_{xid}(h-min)$ 番目の書き込みアクセス数とし、 $M_{xid}(h-min) = M_{xid}(h-min+1)$ でないならば、ステップ S95 をスキップして、ステップ S96 に移る。

【0230】

ステップ S96 で、 $TL = NULL$ でない場合、ステップ S97 で、 $TL = TL -xid$ 、 $xid = TL$ の中の最初のトランザクション識別子として、ステッ

プS82に戻る。

【0231】

他方、ステップS96で、 $TL = NULL$ である場合、ステップS98で、 $e > h - min$ のS-Pointの効果値-e1であるならば、ステップS99で、 $h - min$ 番目のS-Pointを設定して、処理を終了する。また、ステップS98で、 $e > h - min$ のS-Pointの効果値-e1でないならば、ステップS100で、S-Pointは設定せずに、処理を終了する。

【0232】

一例として、記録数 $H = 2$ の場合に、図19のTime2、Time3、Time4、Time5の各時点におけるS-Point管理表128の変化を図24に示す。

【0233】

まず、図24の(a)のTime2の時点のS-Pointは、すでに説明したように、図20と同じである。

【0234】

次に、Time3の時点で、S-Point管理表128にある最後のS-Point番号は2であり、記録数 H と同じであるので、新しいS-Pointを設定するか否か判断する。新しいS-Pointの設定で増える効果値 e は、トランザクション3の $WS_{T3}(1)$ の書込みアクセス数の2である。1番目のS-Pointを削除すると、トランザクションT2の $WS_{T2}(1)$ の書込みアクセス数は2番目のS-Pointの効果値に足されるので、減る効果値は0である(2番目のS-Pointを削除すると減る効果値も同様に0である)。したがって、Time1で設定された1番目のS-Pointは取り消されて新しいS-Pointが設定され、図24の(b)のS-Point管理表のように変わる。

【0235】

Time4の時点で新しいS-Pointの設定で増える効果値 e は、トランザクションT2の $WS_{T3}(2)$ の書込みアクセス数の1である。1番目のS-Pointを削除すると、トランザクションT1の $WS_{T1}(1)$ の書込みアクセス

数+トランザクションT1のWS_{T2}(1)の書込みアクセス数(=4)は2番目S-Pointの効果値に足されるので、減る効果値はこの場合も0である(2番目のS-Pointを削除すると減る効果値も同様に0である)。したがって、Time3で設定された1番目のS-Pointは取り消されて新しいS-Pointが設定され、図24の(c)のS-Point管理表のように変わる。

【0236】

最後に、Time5の時点で新しいS-Pointの設定で増える効果値eは、トランザクション2のWS_{T2}(2)の書込みアクセス数の2である。1番目のS-Pointを削除すると減る効果値は、トランザクションT3のWS_{T3}(1)の書込みアクセス数(=2)である。一方、2番目のS-Pointを削除すると減る効果値は0である。したがって、Time4で設定された2番目のS-Pointは取り消されて新しいS-Pointが設定され、図24の(d)のS-Point管理表のように変わる。

【0237】

(4) トランザクションを中断後に再開するときの処理手順

トランザクションを中断後に再開するときの処理は、第1の構成例のリソースマネージャ12の処理と同じである。

【0238】

(5) トランザクションをコミットするときの処理手順

トランザクションTidをコミットして終了するときには、第1の構成例のリソースマネージャ12が行う手順と同様に、そのトランザクションが行ったデータの更新結果をファイルおよび他のトランザクションのドキュメントに反映させるために、D(Tid)をD-stと他のトランザクションのドキュメントDにマージする処理と、トランザクション待ちグラフ122を調べてそのトランザクションの終了を待ちながら中断している他のトランザクションを再開させる処理とを行う。また、トランザクションリストとトランザクション待ちグラフ122からTidを削除し、トランザクションアクセス系列AS(Tid)とドキュメントD(Tid)も削除する。

【0239】

その他に、S-P o i n t 管理表 1 2 8 からトランザクション T i d の W S 欄を削除して、それにともなう効果値の変更を行う。各 S-P o i n t エントリーにおいて効果値に足されているトランザクション T i d の W S 番目の書込みアクセスリストの書込みアクセス数を引けばよい。

【0240】

(6) トランザクションをアボートするときの処理手順

トランザクション T i d をアボートして終了するときは、第 1 の構成例のリソースマネージャ 1 2 が行う手順と同様に、そのトランザクションが行ったデータの更新結果を破棄するためにドキュメント D-a l l を再作成する処理とトランザクション待ちグラフ 1 2 2 を調べて、そのトランザクションの終了を待ちながら待機している他のトランザクションを再開させる処理を行う。また、トランザクションリストとトランザクション待ちグラフ 1 2 2 から T i d を削除し、トランザクションアクセス系列 A S (T i d) とドキュメント D (T i d) も削除する。ドキュメント D-a l l を再作成は、ドキュメント D-s t にトランザクションリストにあるすべてのトランザクションのドキュメント D を重ねてマージすることで行う。

【0241】

その他に、トランザクションのコミット時と同様に、S-P o i n t 管理表 1 2 8 の変更として、トランザクション T i d の W S 欄の削除とそれにともなう効果値の変更を行う。S-P o i n t 管理表 1 2 8 は効果値の高い D-a l l の状態を記録管理するためのものなので、アボート時に最適な S-P o i n t 設定のスケジュールを決定してそれに従って D-a l l の再作成を行うように実施することもできる。

【0242】

なお、以上の各機能は、ソフトウェアとして実現可能である。

また、本実施形態は、コンピュータに所定の手段を実行させるための（あるいはコンピュータを所定の手段として機能させるための、あるいはコンピュータに所定の機能を実現させるための）プログラムとして実施することもでき、該プログラムを記録したコンピュータ読取り可能な記録媒体として実施することもでき

る。

【0243】

なお、この発明の実施の形態で例示した構成は一例であって、それ以外の構成を排除する趣旨のものではなく、例示した構成の一部を他のもので置き換えたり、例示した構成の一部を省いたり、例示した構成に別の機能あるいは要素を付加したり、それらを組み合わせたりすることなどによって得られる別の構成も可能である。また、例示した構成と論理的に等価な別の構成、例示した構成と論理的に等価な部分を含む別の構成、例示した構成の要部と論理的に等価な別の構成なども可能である。また、例示した構成と同一もしくは類似の目的を達成する別の構成、例示した構成と同一もしくは類似の効果を奏する別の構成なども可能である。

また、この発明の実施の形態で例示した各種構成部分についての各種バリエーションは、適宜組み合わせて実施することが可能である。

また、この発明の実施の形態は、個別装置としての発明、関連を持つ2以上の装置についての発明、システム全体としての発明、個別装置内部の構成部分についての発明、またはそれらに対応する方法の発明等、種々の観点、段階、概念またはカテゴリに係る発明を包含・内在するものである。

従って、この発明の実施の形態に開示した内容からは、例示した構成に限定されることなく発明を抽出することができるものである。

【0244】

本発明は、上述した実施の形態に限定されるものではなく、その技術的範囲において種々変形して実施することができる。

【0245】

【発明の効果】

本発明によれば、階層型データを複数のトランザクションが並行してアクセスする場合にも、トランザクションの分離性を保証することができる、あるいは、その実行が直列化可能であるように処理の順序を制御することができるようになる。

【図面の簡単な説明】

【図 1】 本発明の一実施形態に係るトランザクション処理システムの構成例を示す図

【図 2】 XMLドキュメントの一例を示す図

【図 3】 XMLドキュメントの一例を示す図

【図 4】 XMLドキュメントの一例を示す図

【図 5】 XMLドキュメントの一例を示す図

【図 6】 トランザクション管理表の一例を示す図

【図 7】 同実施形態に係るリソースマネージャの構成例を示す図

【図 8】 トランザクションリストの一例を示す図

【図 9】 トランザクション待ちグラフの一例を示す図

【図 10】 トランザクションアクセス系列の一例を示す図

【図 11】 同実施形態におけるトランザクションの処理を開始するときの処理手順の一例を示すフローチャート

【図 12】 同実施形態におけるトランザクションが読出しアクセスを要求したときの処理手順の一例を示すフローチャート

【図 13】 同実施形態における関数 Eval の処理の一例を示すフローチャート

【図 14】 同実施形態におけるトランザクションが書込みアクセスを要求したときの処理手順の一例を示すフローチャート

【図 15】 トランザクションアクセス系列の一例を示す図

【図 16】 並行処理中のトランザクションの一例を示す図

【図 17】 同実施形態に係るリソースマネージャの他の構成例を示す図

【図 18】 トランザクションアクセス系列の一例を示す図

【図 19】 並行処理中のトランザクションの一例を示す図

【図 20】 S-Point 管理表の一例を示す図

【図 21】 S-Point 管理表の一例を示す図

【図 22】 同実施形態におけるトランザクションが書込みアクセス W を要求したときの WR アクセス衝突の調査の処理手順の一例を示すフローチャート

【図 23】 同実施形態における S-Point 設定の処理手順の一例を示す

フローチャート

【図 2 4】 S - P o i n t 管理表の一例を示す図

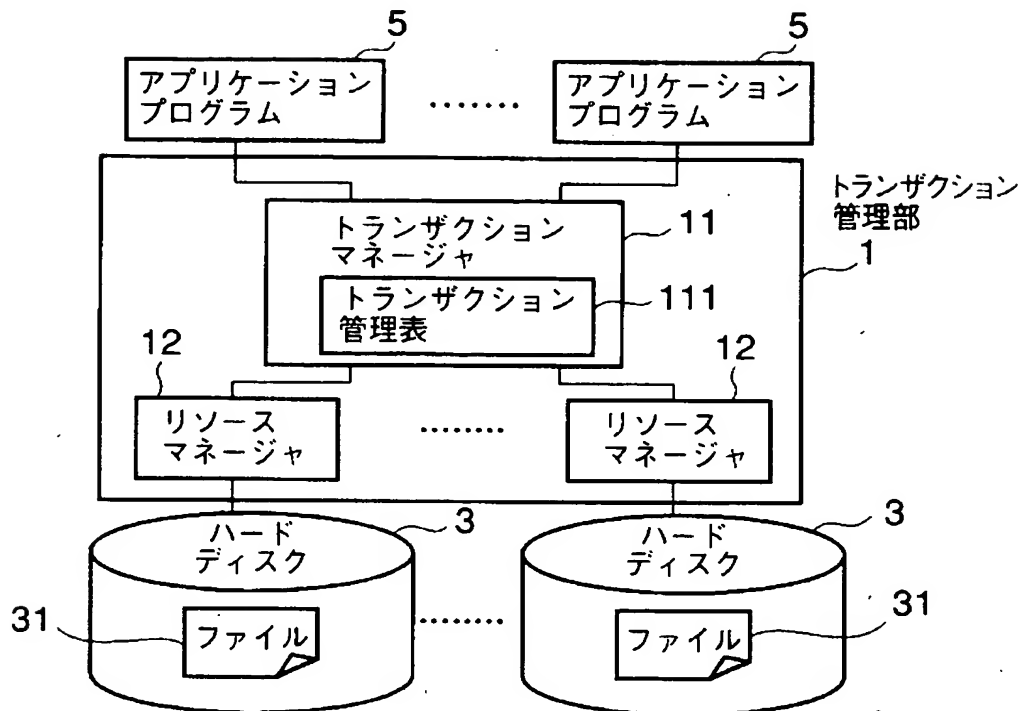
【符号の説明】

1…トランザクション管理部、1 1…トランザクションマネージャ、1 1 1…トランザクション管理表、1 2…リソースマネージャ、3…ハードディスク、3 1…ファイル、5…アプリケーションプログラム、1 2 1, 1 2 5, 1 2 7…ドキュメント、1 2 2…トランザクション待ちグラフ、1 2 3…トランザクションリスト、1 2 4…トランザクションアクセス系列、1 2 6…記録数、1 2 8…S - P o i n t 管理表

【書類名】

図面

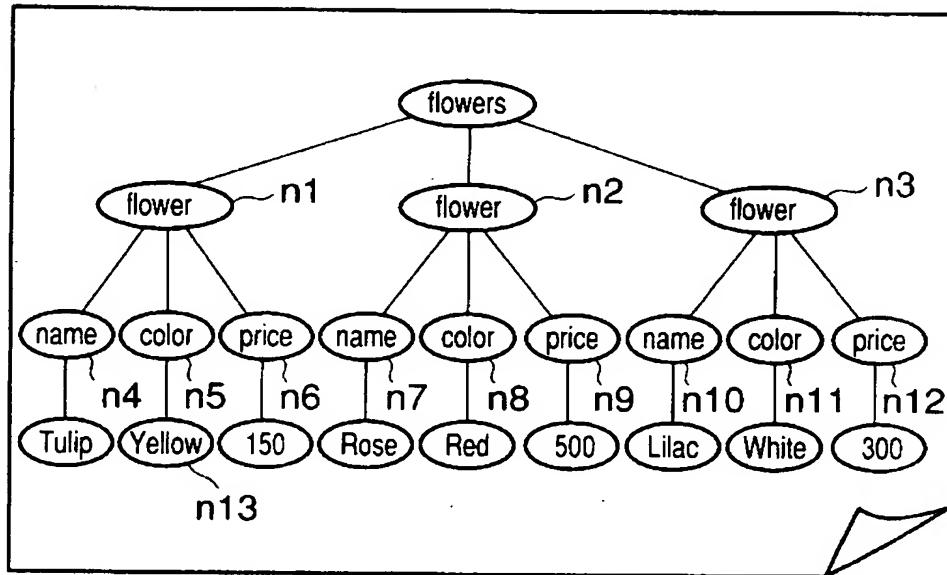
【図 1】



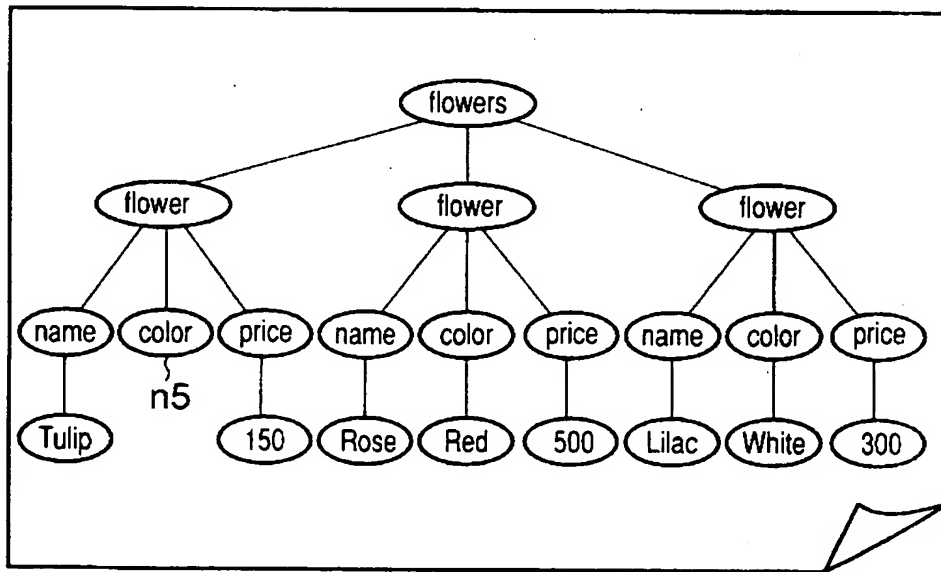
【図 2】

```
<?xml version="1.0"?>
<flowers>
  <flower>
    <name>Tulip</name>
    <color>Yellow</color>
    <price>150</price>
  </flower>
  <flower>
    <name>Rose</name>
    <color>Red</color>
    <price>500</price>
  </flower>
  <flower>
    <name>Lilac</name>
    <color>White</color>
    <price>300</price>
  </flower>
</flowers>
```

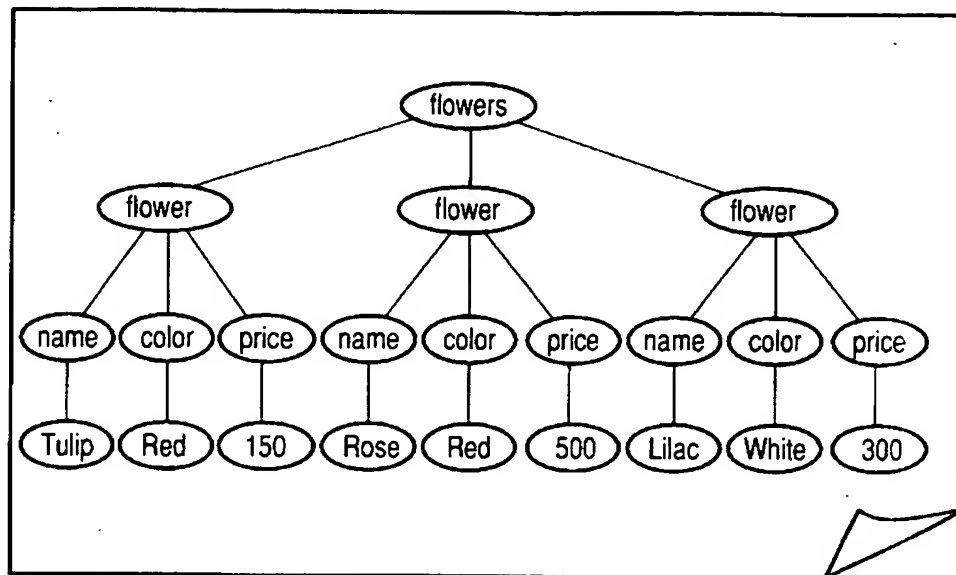
【図 3】



【図 4】



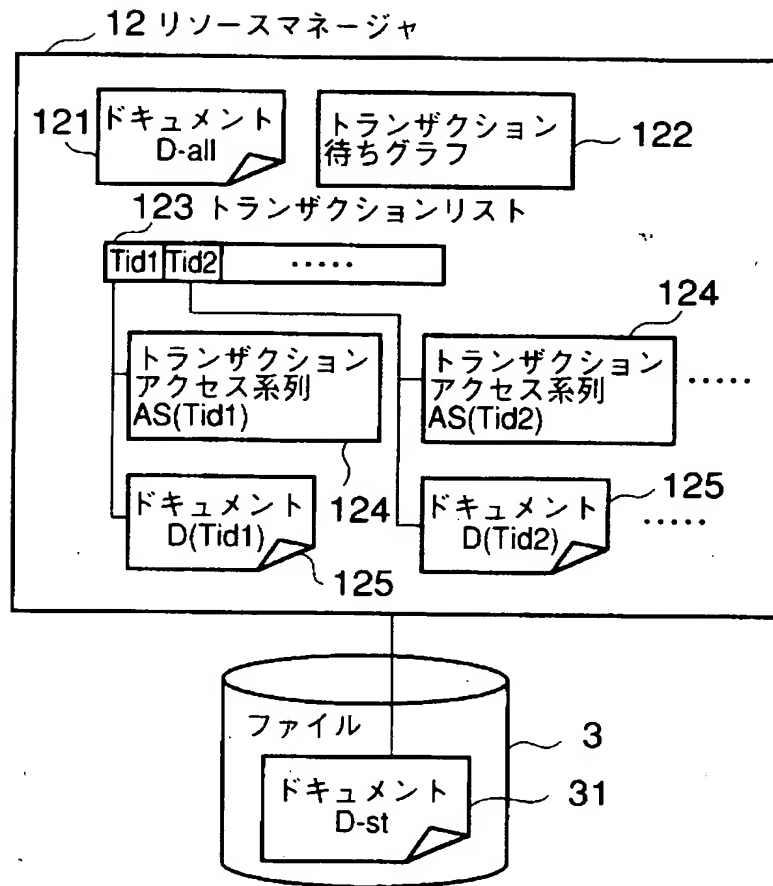
【図 5】



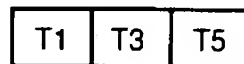
【図 6】

トランザクション	リソースマネージャ
T1	R1
T2	R2
T3	R1,R2
T4	R2
T5	R1

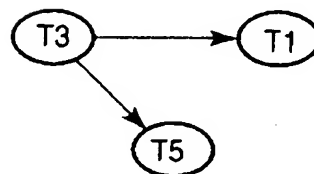
【図 7】



【図 8】



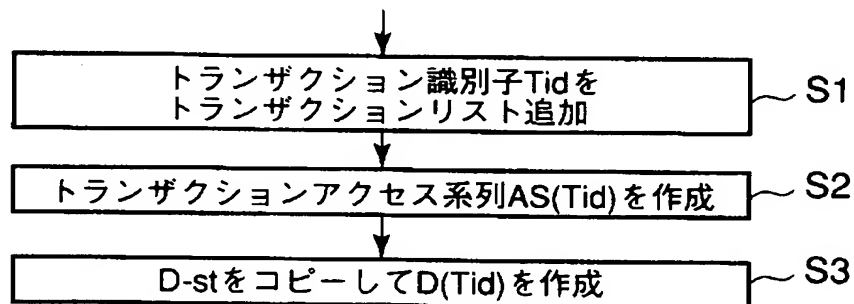
【図 9】



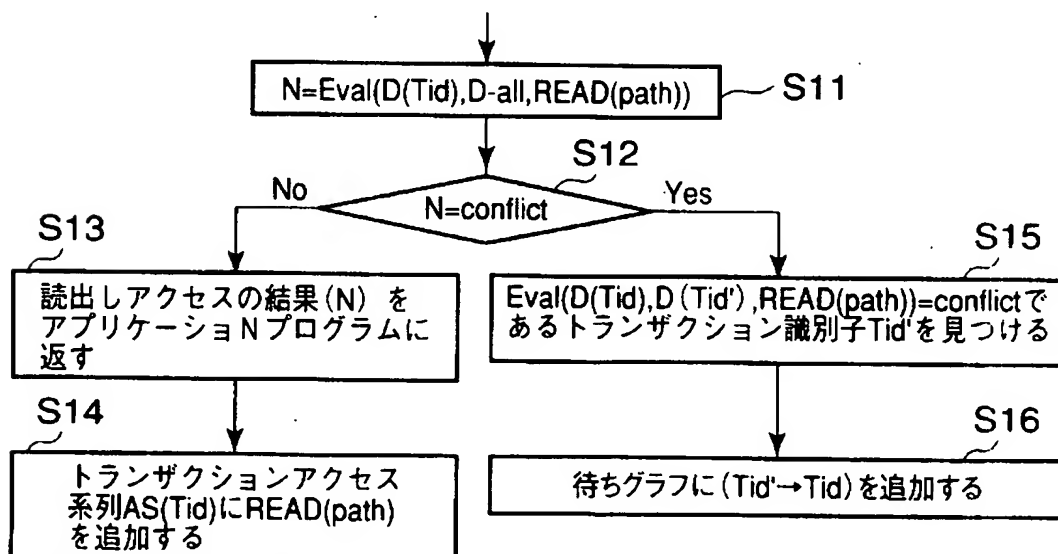
【図10】

アクセス番号	読出しor き込み	アクセス操作
1	r	READ("flower/name")
2	r	READ("flower[name=Tulip]/color")
3	w	REPLACE(node ₂ , "Red")

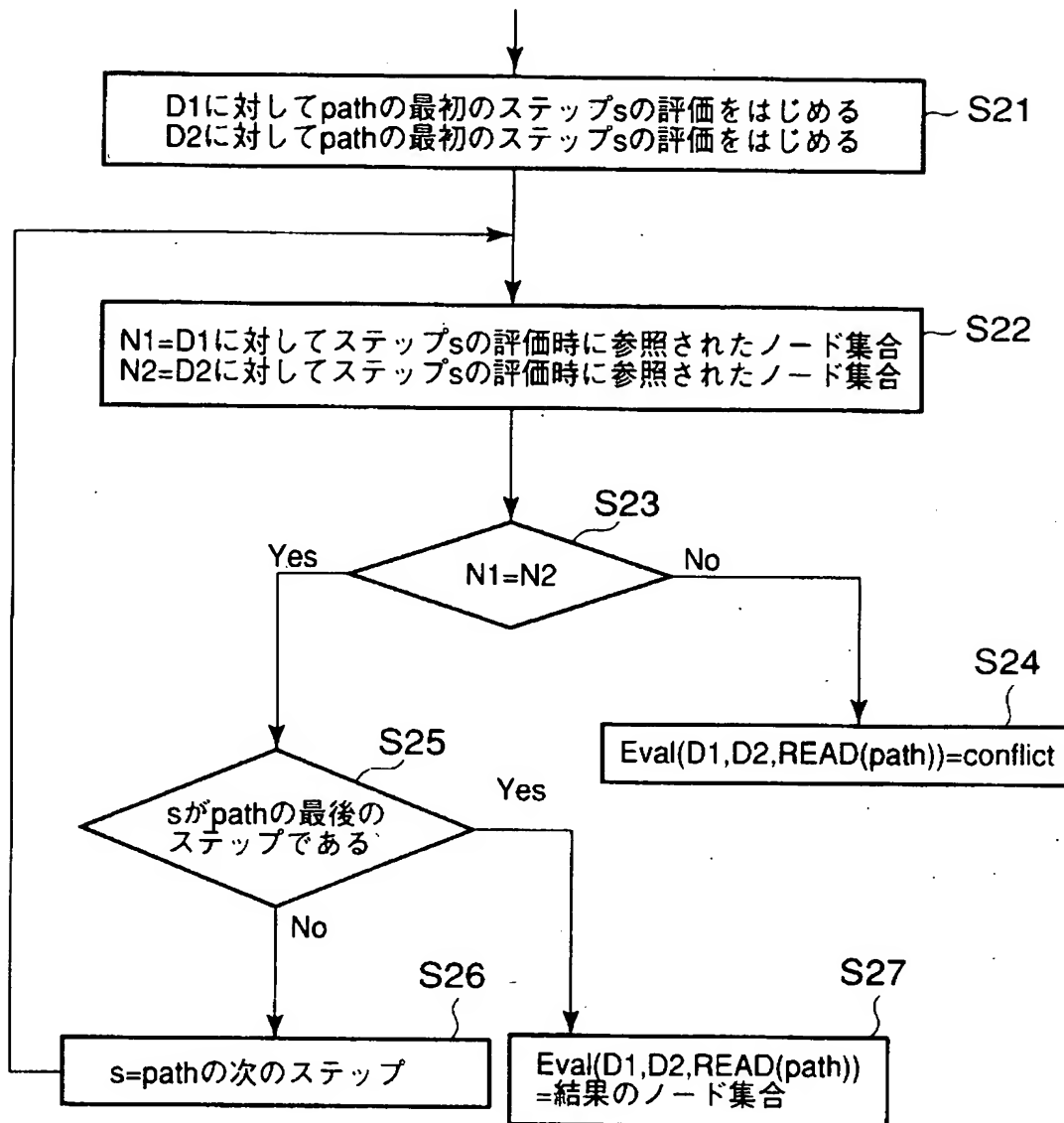
【図11】



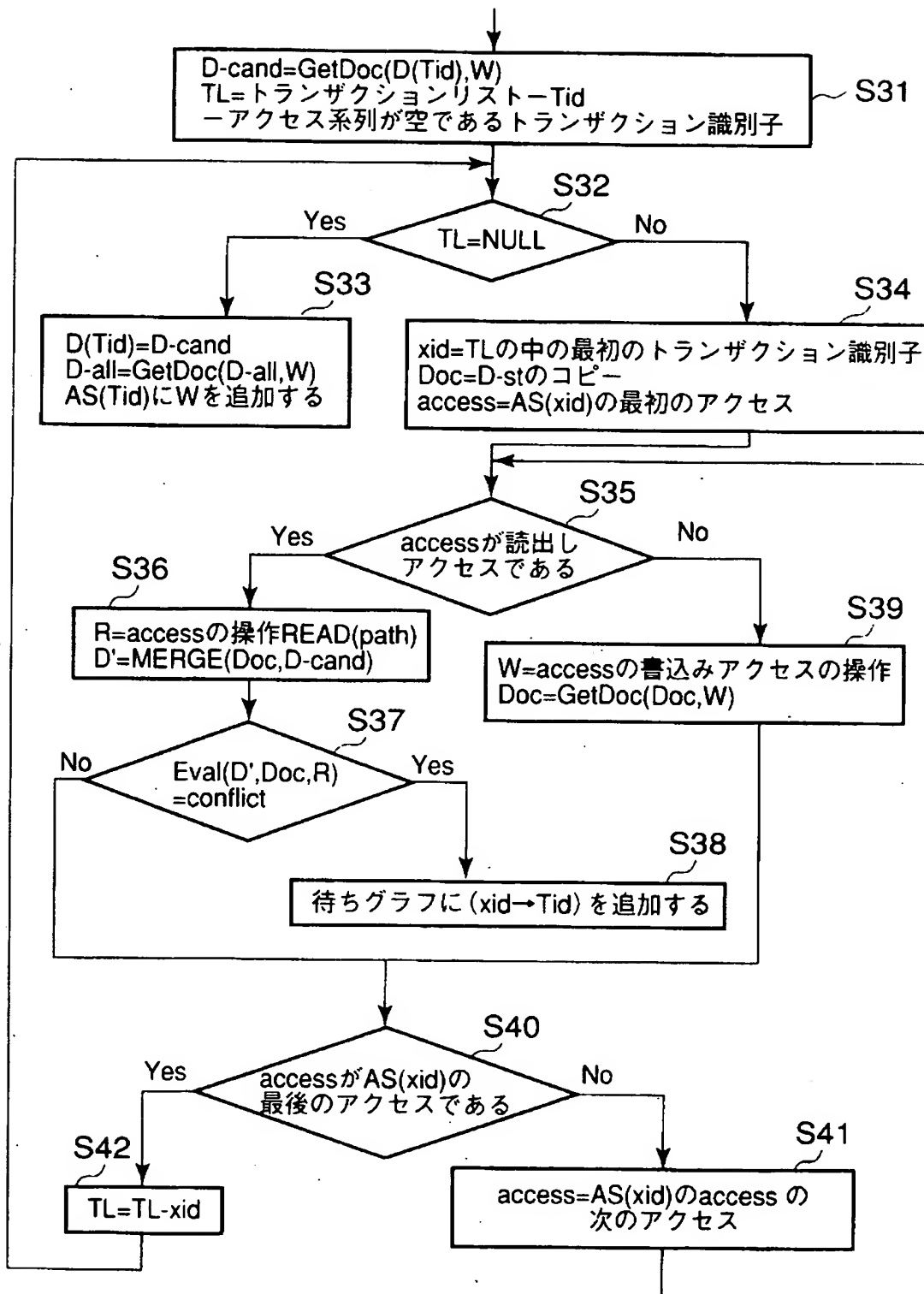
【図12】



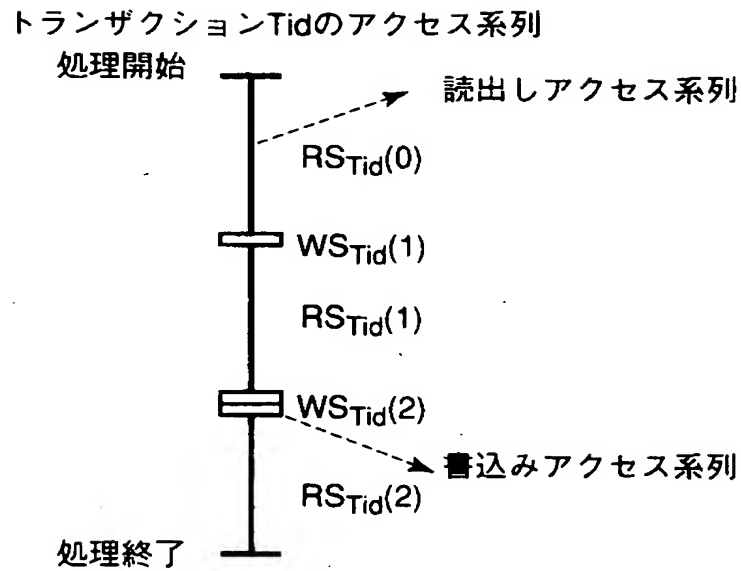
【図 13】



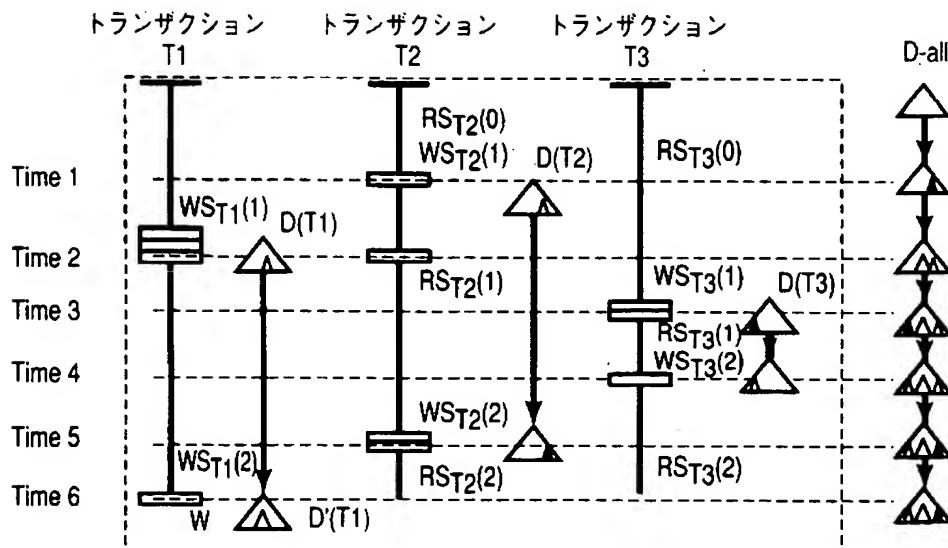
【図 14】



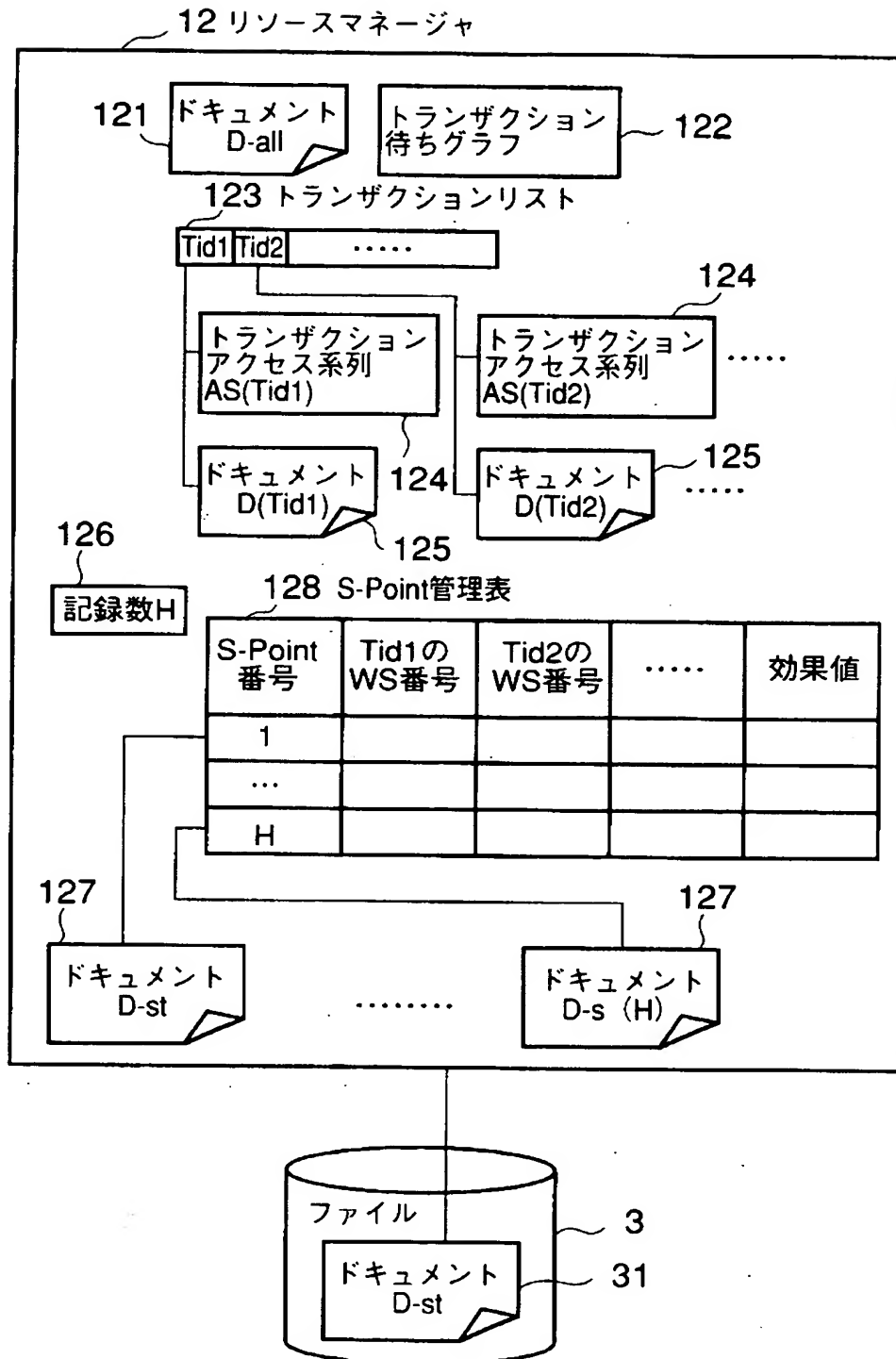
【図 15】



【図 16】



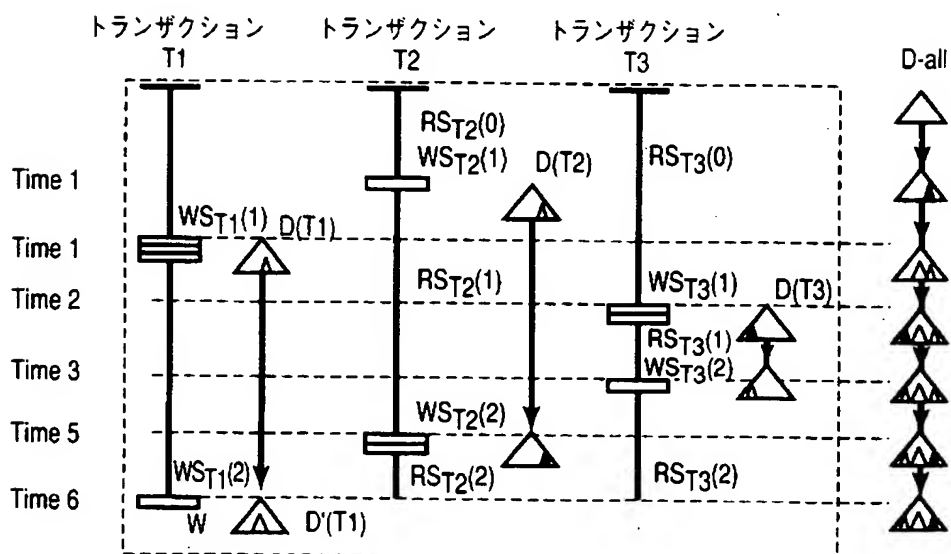
【図 17】



【図 18】

	込み アクセス数	
RS(0)		読出しアクセス操作 ₁ 読出しアクセス操作 ₂
WS(1)	1	書込みアクセス操作
RS(1)		読出しアクセス操作 ₁ 読出しアクセス操作 ₂
WS(2)	2	書込みアクセス操作 ₁ 書込みアクセス操作 ₂
RS(2)		読出しアクセス操作 ₁ 読出しアクセス操作 ₂

【図 19】



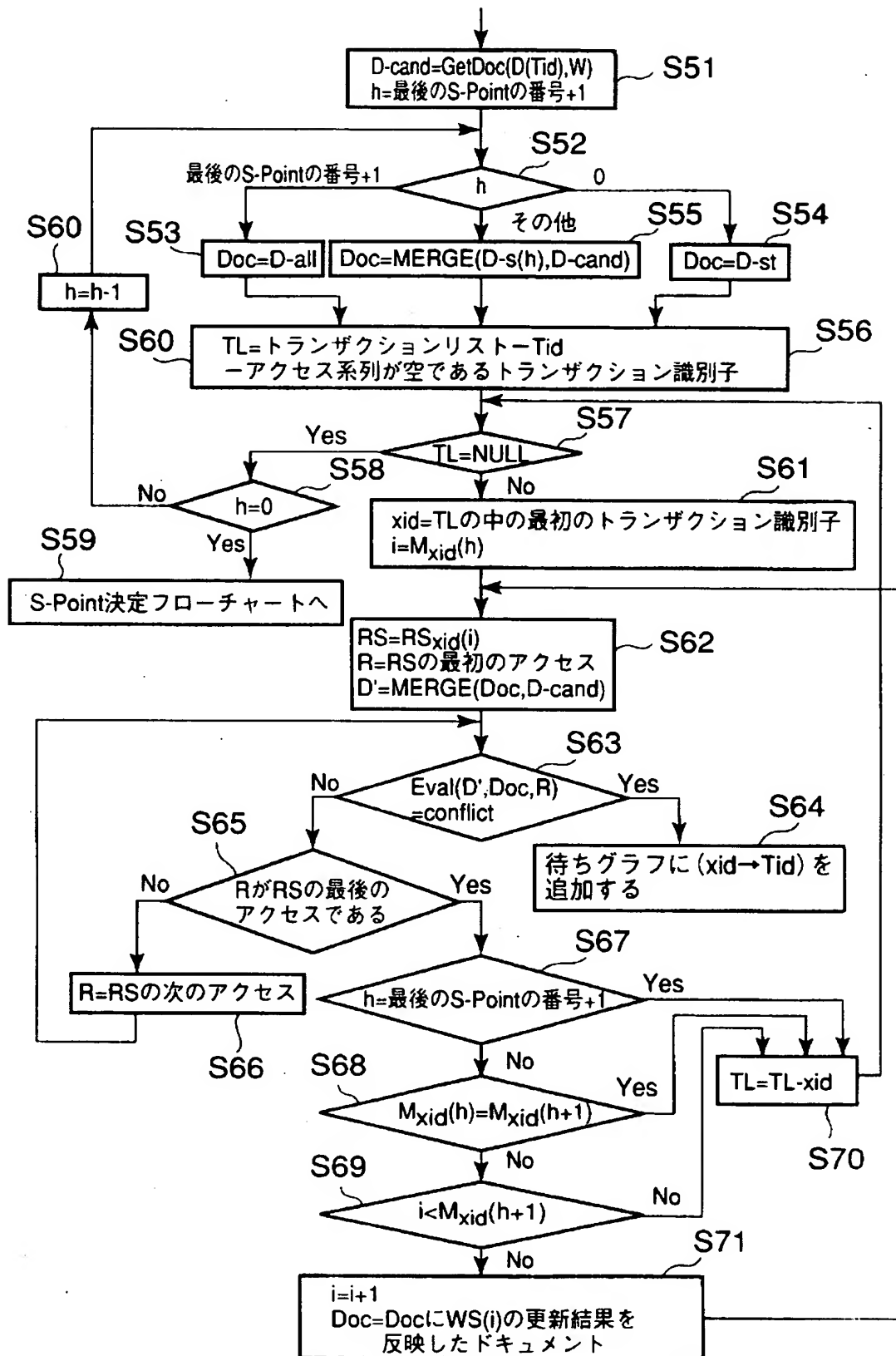
【図 20】

S-Point番号	WS番号			効果値
	T1	T2	T3	
1	0	1	0	1
2	1	1	0	3

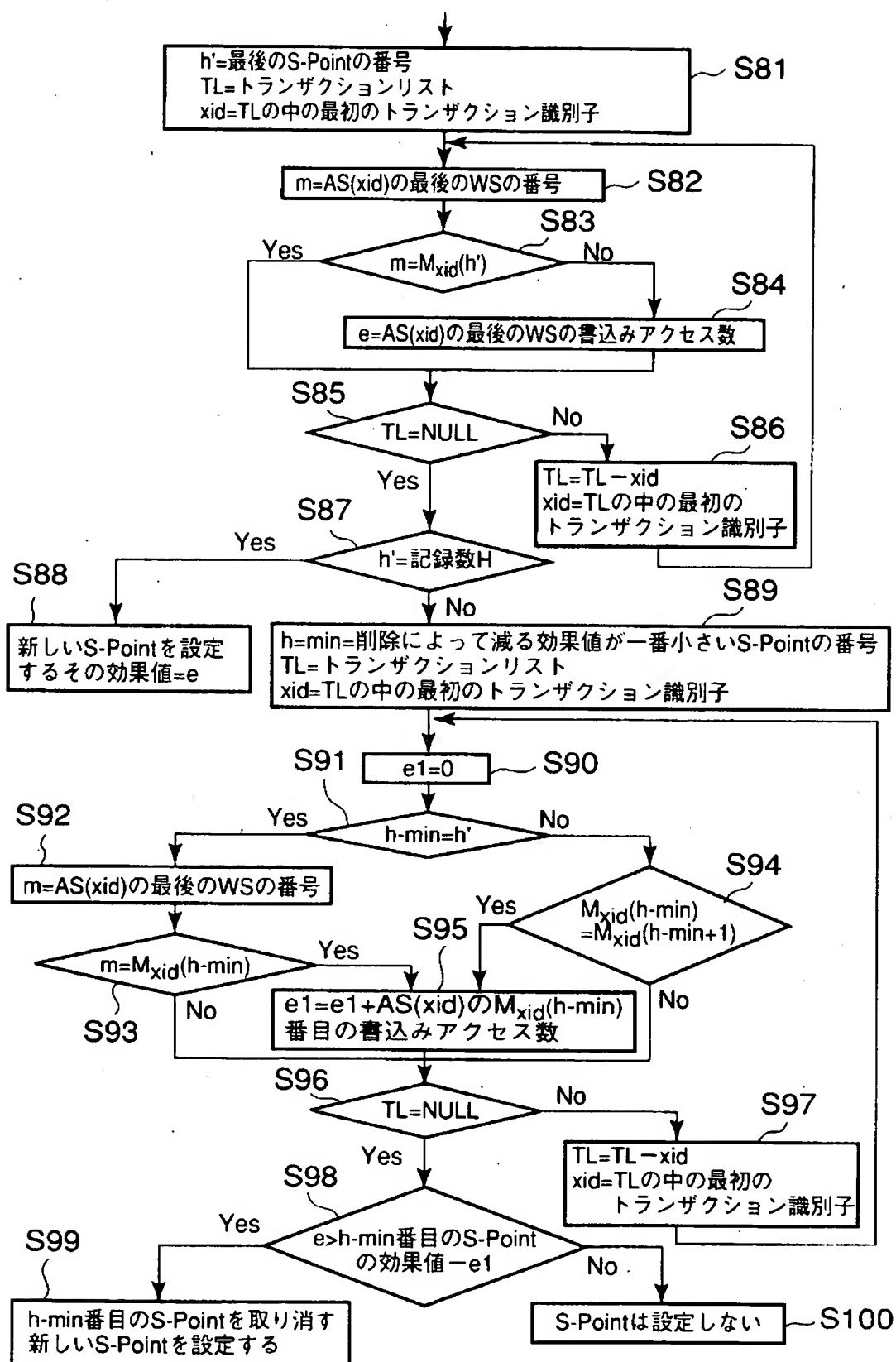
【図 21】

S-Point番号	WS番号			効果値
	T1	T2	T3	
1	1	1	0	4

【図 22】



【図 23】



【図 24】

(a) Time2の時点

S-Point番号	WS番号			効果値
	T1	T2	T3	
1	0	1	0	1
2	1	1	0	3

(b) Time3の時点

S-Point番号	WS番号			効果値
	T1	T2	T3	
+	0	+	0	+
1	1	1	0	4
2	1	1	1	2

(c) Time4の時点

S-Point番号	WS番号			効果値
	T1	T2	T3	
+	+	+	0	4
1	1	1	1	6
2	1	1	2	1

(d) Time5の時点

S-Point番号	WS番号			効果値
	T1	T2	T3	
1	1	1	1	6
2	+	+	2	+
2	1	2	2	3

【書類名】 要約書

【要約】

【課題】 階層型データを複数のトランザクションが並行してアクセスする場合にもトランザクションの分離性を保証することができるトランザクション処理システムを提供すること。

【解決手段】 各トランザクションの開始時に各トランザクション用の階層型データのコピーを作成する。第1のトランザクションがコピーに対して読み出し又は書き込みを行う場合に、該アクセスと該第2のトランザクションがコピーに対して行った書き込み又は読み出しアクセスとが衝突するか判定し、衝突回避を行う。その後、第1のトランザクションが正常に終了する場合には、それがコピーに行った書き込みアクセスを、階層型データに反映させるとともに、第2のトランザクションが未終了ならば該トランザクション用のコピーにも反映させる。

【選択図】 図1

特願 2003-025164

出願人履歴情報

識別番号

[000003078]

1. 変更年月日

2001年 7月 2日

[変更理由]

住所変更

住 所

東京都港区芝浦一丁目1番1号

氏 名

株式会社東芝

2. 変更年月日

2003年 5月 9日

[変更理由]

名称変更

住所変更

住 所

東京都港区芝浦一丁目1番1号

氏 名

株式会社東芝